VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Technologie HTML 5 a .NET MVC 3 pro webové aplikace

HTML 5 and .NET MVC 3 Technology for Web Applications

2013

Martin Haščák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Haščák**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Technologie HTML 5 a .NET MVC 3 pro webové aplikace
HTML 5 and .NET MVC 3 Technology for Web Applications

Zásady pro vypracování:

Cílem práce je zachytit nové koncepty HTML 5 ve vazbě na možnosti platformy .NET MVC 3.

1. Zmapujte a popište hlavní prvky HTML 5 a navazující technologie.
2. Zmapujte a popište koncept .NET MVC 3, a to s ohledem na vývoj v rámci HTML 5.
3. Navrhněte ukázkovou aplikaci, která bude ilustrovat klíčové možnosti kombinace obou technologií. V rámci ukázkové aplikace se zaměřte na implementaci řešení, které bude vhodné pro správu a provoz online kurzů.
4. Implementujte navrženou aplikaci s ohledem na správné využití MVC 3 přístupů.
5. Zhodnoťte možnosti využití MVC 3 při vývoji nových aplikací v rámci HTML 5.

Seznam doporučené odborné literatury:

[1] Brian P. Hogan: HTML5 and CSS3: Develop with Tomorrow's Standards Today, Pragmatic Bookshelf, ISBN: 978-1934356685, 2011
[2] Jon Galloway a kolektiv: Professional ASP.NET MVC 3, Wrox, ISBN: 978-1118076583, 2011

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Michal Radecký, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2013 ..... Havlíček Martin

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013 ..... Havlíček Martin

Na tomto místě bych chtěl poděkovat vedoucímu své bakalářské práce,
panu Ing. Michalu Radeckému, Ph.D. za poskytnuté rady, trpělivost a připomínky k obsahu a formě zpracování práce.

Abstrakt

Tato práce se věnuje aktuálním trendům ve vývoji webových aplikací. Konkrétně se zaměřuje na technologie HTML 5 a ASP.NET MVC 4, jejichž klíčové prvky jsou popsány v prvních dvou kapitolách. Vzájemné propojení těchto technologií je ilustrováno na ukázkové aplikaci pro správu a provoz online sdílených tabulí. Hlavním cílem aplikace je umožnit komunikaci v reálném čase, mezi uživateli kurzů, prostřednictvím kreslícího plátna a diskuse. Práce v závěru seznámí čtenáře s problémy, které provázely samotný vývoj, nabídne možné řešení a zhodnotí, zda použité technologie mají budoucnost.

Klíčová slova: HTML 5, ASP.NET MVC, aplikace, web

Abstract

This bachelor thesis deals with topical trends pertaining to development of website pages. It particularly focuses on technology HTML 5 and MVC 4, whose key terms are described in first two chapters. Subsequently, the interconnection between them is illustrated in an example application for management and operation of online-shared whiteboards. The major purpose of the application is to enable communication among the users of courses in real time, which should be accomplished via both online drawing canvas and online discussion. At last, it outlines difficulties related to the development itself and it also offers possible solutions and tries to evaluate whether used technologies have some potential to succeed.

Keywords: HTML 5, ASP.NET MVC, application, web

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
API	–	Application Programming Interface
ASPX	–	Active Server Page Extended File
CLOUD	–	Cosmics Leaving Outdoor Droplets
CLR	–	Common Language Runtime
CRSF	–	Cross-site Request Forgery
CSS	–	Cascading Style Sheets
GUI	–	Graphical user interface
HTTP	–	Hypertext Transfer Protocol
HTML	–	Hyper Text Markup Language
IIS	–	Internet Information Services
IE	–	Internet Explorer
JSON	–	JavaScript Object Notation
MVC	–	Model-View-Controller
ORM	–	Objektově relační mapování
REST	–	Representational State Transfer
RIA	–	Rich Internet Application
SQL	–	Structured Query Language
SVG	–	Scalable Vector Graphics
T4	–	Text Template Transformation Toolkit
URL	–	Uniform Resource Locator
VPS	–	Virtuální privátní server
WHATWG	–	The Web Hypertext Application Technology Working Group
WWW	–	World Wide Web
W3C	–	World Wide Web Consortium
XHR	–	XMLHttpRequest

Obsah

Seznam obrázků

1 Úvod

Technologický vývoj, jakožto i vývoj samotného internetu, jde rychle kupředu. Internet je stále rychlejší a hardware dokonalejší. S příchodem dotykových telefonů, tabletů a dalších chytrých zařízení s možností připojení k internetu téměř odkudkoliv, jsou v posledních letech kladeny vyšší nároky na funkčnost i rychlost webových aplikací. S tímto rychlým vývojem roste i průmyslová poptávka po modernizaci technologického zázemí pro vývoj webových aplikací, které se snaží svým chováním stále více přiblížit aplikacím desktopovým.

Cílem této práce je popsat, a prakticky prezentovat, inovativní prvky v rámci nového standardu HTML 5 v kombinaci s technologií ASP.NET MVC pro vývoj webových aplikací. Vzhledem k tomu, že v průběhu tvorby bakalářské práce byla uvolněna nová verze frameworku MVC 4, se práce bude věnovat novější verzi, byť je v zadání uvedena verze MVC 3. Výstupem této práce bude ukázková aplikace, která názorně demonstruje některé vybrané prvky z obou technologií.

2 HTML 5

HTML 5 je nově vznikající standard jazyka HTML pro tvorbu webových stránek. Vychází z jazyka HTML 4 a XHTML 1, s nimiž je i zpětně kompatibilní.[1] Základy tohoto nového standardu položila skupina WHATWG v roce 2004.[1] Rozšířila původní HTML 4 o nové ovládací prvky, které zjednodušovaly validaci dat formulářů na klientské straně pod názvem Web Forms 2.0.[1] Do vývoje se později po vzájemné dohodě zapojilo i konsorcium W3C, které přijalo koncept definovaný skupinou WHATWG a vytvořily pro něj označení HTML 5.[2][1] Hlavní změnou oproti jeho předchůdcům je fakt, že se nejedná pouze o značkovací jazyk, nýbrž o soubor webových technologií, z něhož nové HTML tvoří jen jednu část, která přináší nové strukturální elementy, syntaxi, sémantiku atd.

2.1 Současný stav a budoucnost

Aktuálně se HTML 5 jeví jako stabilní, jelikož ho většina výrobců prohlížečů již implementovala do jejich nejnovějších verzí. Často však lze narazit na fakt, že různé druhy prohlížečů podporují pouze určitou část dostupné funkčnosti. Vznikají tak tabulky s porovnáním, které verze prohlížečů implementují konkrétní funkčnosti jazyka. Konsorcium W3C se snaží zajistit silnou interoperabilitu mezi prohlížeči, proto se může zdát vývoj specifikace HTML 5 poměrně zdlouhavý. Předpokládaný datum dokončení, a vydání finální verze, se odhaduje na konec roku 2014.[3] Stále se však může cokoli změnit. HTML 5 ještě obsahuje množství chyb a API, které jsou v ranním stádiu vývoje.

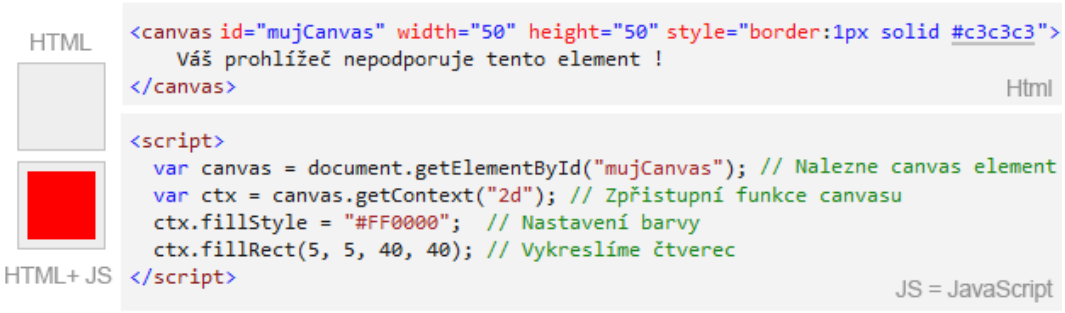
2.2 Prezentační část jazyka

Při tvorbě vzhledu webových stránek bylo často nezbytné stránku rozdělit do více bloků pomocí elementů <*div*> na hlavičku, patičku, navigaci a podobně. Toto často opakované a používané rozdělení vedlo k vytvoření nové sady elementů <*footer*>,<*header*>, <*article*> atd. Tyto elementy v podstatě fungují na stejném principu, ale výrazně zlepšují přehlednost kódu jak pro autora webu, tak pro stroje, které se kód snaží zpracovat a porozumět mu. Syntaxe samotného jazyka se skoro nezměnila. Byla ale definována nová detailně zpracovaná parsovací pravidla, která výrobcům prohlížečů vymezují jasné mantinely, jak mají být jednotlivé elementy použity. [5]

Nová specifikace HTML 5 může výrazně pomoct sjednotit vykreslování webových stránek a odstranit tak dosud přetrvávající problémy s různorodou reprezentací obsahu mezi prohlížeči. Nepopisuje pouze jak zpracovávat správně vytvořený kód, ale definuje i jak se má prohlížeč chovat, když je kód dokumentu vytvořen špatně. Doposud výrobci prohlížečů museli tomuto problému čelit sami, ale definováním jasného postupu zpracování chyb jim dává možnost soustředit se spíše na nové funkce prohlížeče, než vyvíjet řešení vlastní. [5]

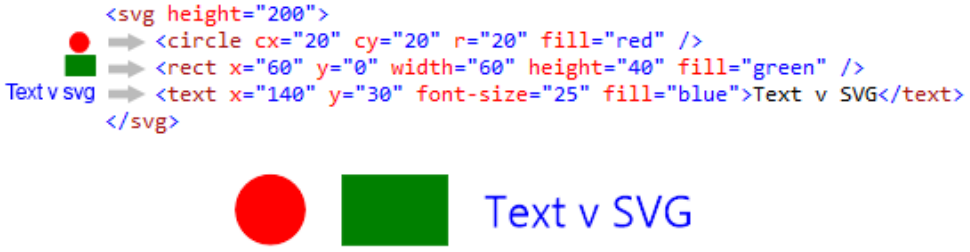
Další způsob, jak ovlivňovat vzhled webu, HTML 5 přináší v podobě kreslícího plátna CANVAS a podpoře formátu SVG. Canvas je nový element jazyka HTML, který

umožňuje pomocí JavaScriptu vykreslovat různé grafické objekty, animace, či herní grafiku přímo za běhu. Použití tohoto elementu a vykreslení jednoduchého grafického tvaru je na obr. 1.



Obrázek 1: Ukázka canvas elementu a jeho API

SVG je na rozdíl od Canvasu značkovací jazyk, který pomocí syntaxe XML umožňuje vykreslovat v rámci webové stránky vektorovou grafiku.[4] Ta je nezávislá na rozlišení. Byl vytvořen a poprvé standardizován konsorciem W3C v roce 2001 jako reakce na průmyslovou poptávku. Aktuálně se hojně používá pro tvorbu GUI nejen webových aplikací. Popis grafiky pomocí XML má výhodu v tom, že ji lze následně jednoduše strojově zpracovat i na mobilních zařízeních. Ukázka aplikace SVG formátu v rámci webu je na obr. 2.[7]



Obrázek 2: Ukázka SVG

Předchůdci HTML 5 neměli v základu podporu multimediálního obsahu. Pro přehrávání videa a zvuku se často používaly různé pluginy, např. (Flash, Silverlight, Java Applety, ...), které bylo nutné instalovat do prohlížeče. Díky novým elementům *audio* a *video* HTML 5 umožňuje vkládat na web multimediální obsah bez vazby na tato rozšíření. Oba elementy mají několik atributů, kterými lze provést jejich základní nastavení včetně možnosti změny vzhledu. Bylo vytvořeno i rozsáhlé API pro ovládání těchto multimediálních prvků. Ukázku vytvoření jednoduchého video a audiopřehrávače ilustruje obr. 3.

Obrázek 3: Ukázka multimediálních elementů

Při použití těchto elementů může nastat problém s podporu kodeků v rámci webových prohlížečů. Snad každý prohlížeč implementuje jiný formát po přehrávání zvuku, ale i videa.[4] Proto je často nutné vkládat multimediální obsah ve více formátech, tak jak je ilustrováno na obr. 3.

Další praktickou novinkou je inovace v rámci formulářových elementů. Některé vývojáře může potěšit implementace automatické validace vstupních dat na straně prohlížeče bez použití JavaScriptu, což bezpochyby tvůrcům webů ušetří čas při tvorbě vlastních validačních pravidel. Výraznou změnou prošel i element *input*, který umožňuje definovat nové typy vstupních dat. Příkladem může být typ *datetime*. Ten zobrazuje automaticky kalendář nebo typ *color*, jenž zobrazí dialog pro výběr barvy.

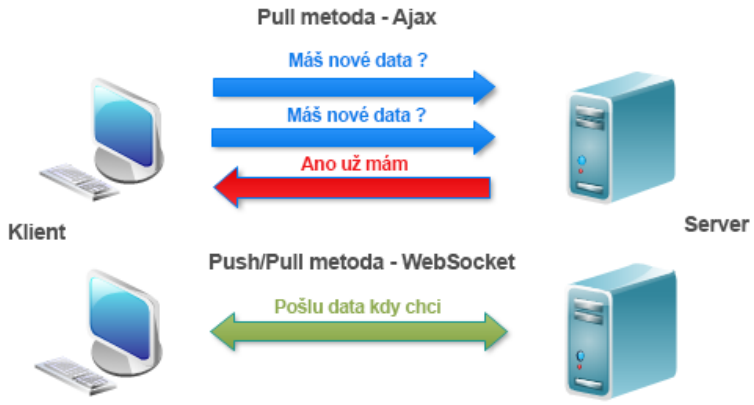
2.3 Funkční část jazyka HTML 5

HTML 5 po funkční stránce přináší řadu inovačních technologií, na které tvůrci webových aplikací nebyli zvyklí nebo je v rámci možností nahrazovali technologiemi jinými. Veškerá nová funkčnost je k dispozici prostřednictvím JavaScriptových API na straně webového prohlížeče. Příkladem může být History API, které řeší časté problémy s používáním tlačítek *zpět* a *vpřed* u aplikací využívajících JavaScript pro změnu obsahu webu bez nutnosti aktualizace stránky. HTML 5 rovněž nabízí API umožňující zpracovávat více paralelních vláken na pozadí prohlížeče bez vlivu na skripty běžící v uživatelském rozhraní webu, podporu Drag and Drop a další zajímavé technologie.

2.3.1 Komunikace

V oblasti komunikace se lze setkat s pojmy jako je Web Socket nebo SSE (Server-Sent Events). Obě technologie zprostředkovávají komunikaci se severem metodou klient-server. Rozdíl však nastává ve směru samotné komunikace a použitém protokolu. Web Sockets API umožňuje navázání obousměrné komunikace mezi klientem (prohlížečem) a serverem za použití jednoho TCP kanálu. Dosud se pro komunikaci se serverem musela používat tzv. pull metoda (Ajax). Ta fungovala na asynchronním principu, že pokud klient od serveru něco chce, musí se ho na to zeptat. Web Socket funguje na principu push/pull

komunikace pomocí síťových soketů. Hlavní výhodou je možnost serveru zasílat data klientovi a opačně v rámci jednoho spojení, bez nutnosti dotazování jednotlivých stran. Využití tohoto API může mít budoucnost např. u online her, při hře mezi více hráči nebo k vytvoření chatovací aplikace. Porovnání běžné komunikace s pomocí Web Socket je naznačeno na obr. 4.[7]

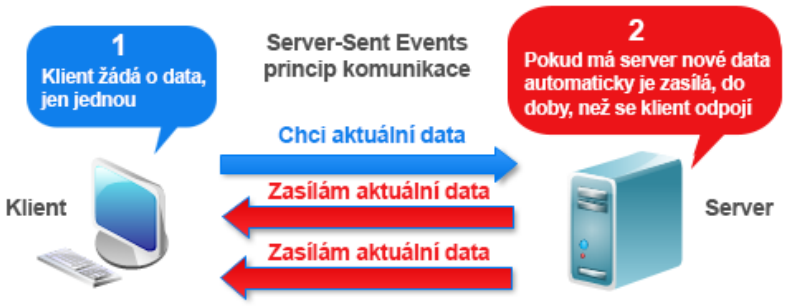


Obrázek 4: Porovnání metod komunikace u Web Socket a Ajax

U SSE, na rozdíl od Web Socket, probíhá přímá komunikace pouze v jednom směru, a to od serveru ke klientovi. Rovněž bez nutnosti opakovaného dotazování se serveru o aktuální data. Komunikace probíhá na pozadí, zasíláním zpráv pomocí HTTP protokolu. Zprávy využívají UTF-8 kódování, přičemž jsou vždy navzájem odděleny novým řádkem. Použití HTTP protokolu přináší vyšší režijní nároky oproti Web Socket protokolu, odpadá však nutnost tvorby vlastního webového serveru. SSE poskytuje řadu zajímavých doplňkových funkcí, např. možnost automatického obnovení spojení nebo číslování zpráv. To umožňuje, v případě výpadku spojení na základě posledního *id* zprávy poskytnutého klientem, zaslat všechny nedoručené zprávy. SSE je vhodný pro méně náročnější aplikace na rychlost spojení a především u aplikací, kde není potřeba obousměrná komunikace. Velmi názorným příkladem může být automaticky aktualizovaná nástěnka Facebooku či Twitteru. Zajímavé využití může být i monitorování zatížení serveru v reálném čase. Princip SSE ilustruje obrázek 5.[8]

2.3.2 Webové úložiště

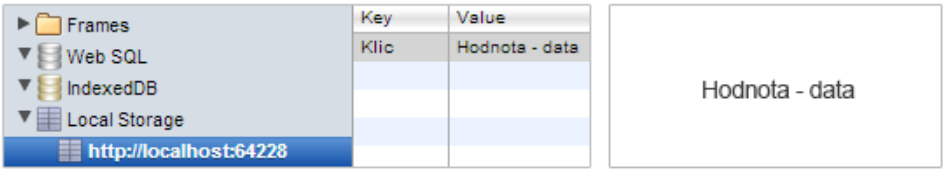
Doposud se pro uložení dat v rámci webových aplikací používaly různé databázové servery, případně cookies prohlížeče, jejichž úložný prostor je omezen na pouhé 4 kB. Uchovávat stav, např. uživatelského rozhraní aplikace tak, jak ho klient při své poslední úpravě zanechal, by vyžadovalo často aktualizovat data v databázi a klást tak mnoho požadavků na server. Pomocí Local Storage API a jiných HTML 5 novinek je možné tato data ukládat na straně klienta. Kapacita, kterou Local Storage nabízí, je v roli výrobců webových prohlížečů. Každý poskytuje jinou velikost možného prostoru, obvykle



Obrázek 5: Princip komunikace SSE

od 2 do 10 MB.[6] Data se u klienta ukládají v páru klíč-hodnota. Použití tohoto API v praxi je velmi snadné, viz. ukázka na obr. 6. Záznamy v lokálním úložišti je možné zobrazit i editovat např. v prohlížeči Chrome, který má integrovaný nástroj pro webové vývojáře.[9]

```
<div id="edit"></div>
<script>
    // Uloží data do lokálního uložiště
    localStorage.setItem('Klic', "Hodnota - data");
    // Nelezne element pod ID=edit a vloží do něj obsah z lokálního uložiště
    document.getElementById("edit").innerHTML = localStorage.getItem('Klic');
</script>
```



Obrázek 6: Ukázka použití Local Storage API

HTML 5 a jeho Off-line API nyní poskytuje řešení pro tvorbu aplikací, které bude možné částečně dále používat i v případě výpadku internetového připojení. Prvky, dostupné v off-line režimu, se definují v tzv. manifestu. Jedná se především o obrázky, HTML, JavaScriptové, CSS a další soubory, jež se na základě manifestu uloží na straně klienta. JavaScriptové API pak zajistí automatickou synchronizaci prvků manifestu při aktualizaci obsahu na straně serveru. Data v mezipaměťi (cache) prohlížeče jsou tedy v případě ztráty spojení stále aktuální. Manifest je normální textový soubor s jasně definovanou strukturou. Odkaz na tento soubor se definuje pomocí atributu *manifest* elementu HTML. Demonstrace manifestu a jeho aplikace je na obr. 7. Manifest je celkem rozdělen na tři části. Ze všeho nejdříve je ale potřeba deklarovat, že se jedná o soubor manifestu pomocí příkazu *CACHE MANIFEST*. V části *CACHE* se definují soubory, které se uloží

lokálně na straně klienta. Část *FALLBACK* slouží k přesměrování na alternativní zdroj v případě, že není požadovaný prvek nalezen. Poslední část *NETWORK* slouží k definování prvků, které nemohou být uloženy a nebudou z mezipaměťi dostupné.[11]

```
<html manifest="/Home/Manifest">

    CACHE MANIFEST
    # Version 1.5
    CACHE:
    /home/index
    /Scripts/mainWeb.js
    /Images/logo.png
    /Images/pozadi.png
    FALLBACK:
    / /home/offline
    NETWORK:
    *
```

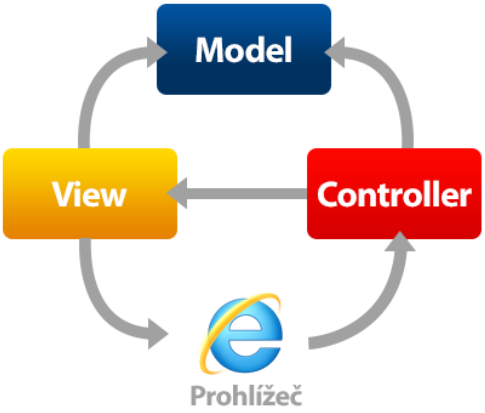
Obrázek 7: Ukázka manifest souboru

Posledním typem úložiště, které HTML 5 nabízí je Indexed Database API. Umožňuje ukládat na straně klienta jednoduchou hierarchii objektů a nad nimi použít rychlé vyhledávání pomocí indexů. Velikost jednoho řádku v DB není omezena.[10] Celková kapacita databáze se ale může lišit u každého prohlížeče.[10] Zatím je tato specifikace v raním stádiu vývoje, může se tak časem cokoli změnit. Některé majoritní prohlížeče však již toto API podporují viz. obr. 6, kde je ilustrována jeho podpora v prohlížeči Chrome.[7]

3 ASP.NET MVC 3

3.1 Představení technologie

ASP.NET MVC je framework pro tvorbu dynamických webových stránek a aplikací vyvíjený společností Microsoft pod Open Source licencí. Je postaven na technologii .NET Frameworku, který vývojářům nabízí možnost programovat v libovolném jazyce kompatibilním s CLR. Jedná se zejména o C#, Visual Basic, JScript.NET aj. První verze frameworku vnikla v roce 2009 a každý rok se vydává nová.[14] Aktuální je pod označením ASP. NET MVC 4. Zkratka MVC (Model-View-Controller) z názvu napovídá, že framework implementuje návrhový vzor, který byl poprvé formulován v roce 1979 panem Trygve Reenskaugem.[12] Jedná se o architektonický vzor, jehož hlavním cílem je oddělit uživatelské rozhraní od aplikační logiky a datové vrstvy tak, aby se každá část starala o svoji specifickou úlohu. [12]



Obrázek 8: Princip frameworku

Interakce s uživatelem v rámci frameworku pak probíhá v cyklu, který je naznačen na obr. 8. Uživatel vyvolá akci, ta je zpracovaná frameworkem a ten pomocí Routingu určí, jaký Controller má danou akci zpracovat. Controller pracuje s Modelem, jenž zapouzdřuje aplikační logiku aplikace a přístup k databázi. Důležité je, že sám Model o nikom neví, je ale schopen poskytnout data, která budou zobrazena uživateli. Jakmile Controller zpracuje uživatelský požadavek (např. na základě poskytnutých parametrů, získá pomocí Modelu požadované objekty z databáze), vybere View a ten se postará o jejich zobrazení. View je komponenta představující uživatelské rozhraní aplikace a obvykle je vytvářena na základě dat poskytnutých modelem. Hlavní výhodou při použití ASP.NET MVC je důsledné oddělení aplikační logiky od uživatelského rozhraní, což umožňuje efektivněji testovat kód aplikace v rámci jednotlivých komponent a zajišťuje lepší podporu paralelního vývoje. Každá komponenta frameworku je navržena tak, aby šla jednoduše nahradit nebo upravit. V případě potřeby pak lze vytvořit vlastní ša-

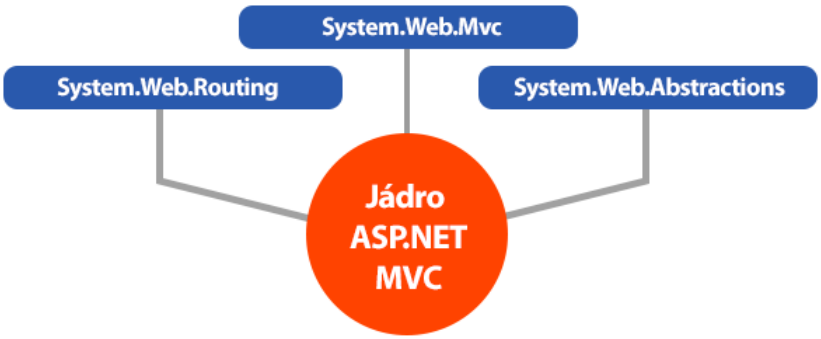
blonovací systém, upravit vestavěné routování, či napsat vlastní továrnu pro vytváření Controllerů.[15]

3.2 Vývojové prostředí

Nástroj, který nabízí Microsoft pro tvorbu ASP.NET MVC aplikací, je Visual Studio. Jedná se o vývojové prostředí nabízející již v základní verzi podporu IntelliSense a předpřipravené šablony projektů. Při vytvoření nového projektu je automaticky vygenerována struktura aplikace a nastaveny všechny potřebné reference na knihovny frameworku. Visual Studio nabízí možnost zvolit si programovací jazyk (C#, Visual Basic) a zobrazovací Engine (Razor, ASPX), který se bude při vývoji využívat. Zároveň umožňuje do vytvořené aplikace jednoduše instalovat knihovny a různé doplňky třetích stran pomocí Nuget Packet Manageru. Jedním z hlavních účelů tohoto balíčkovacího systému je vypořádat se se závislostmi mezi assemblies (česky sestavení) a správou jejich verzí.[21] Při instalaci různých rozšíření může totiž nastat situace, kde daný balíček vyžaduje novější verzi knihovny, než je aktuálně využívána. Balíčkový manažer je tyto závislosti schopen rozpoznat, upozornit na ně a případně aktualizovat potřebné knihovny [21].

3.3 Hlavní části frameworku

Jádro celého MVC Frameworku je složeno ze tří hlavních assembly, viz obr. 9, které rozšiřují jmenný prostor *System.Web*. Assembly je v podstatě soubor tříd, rozhraní a dalších souborů. V .NET Frameworku odpovídá *dll* nebo *exe* souboru.[18] Rovněž zajišťuje správu verzí a bezpečnostních oprávnění.



Obrázek 9: Složení jádra ASP.NET MVC framewoku

Obor názvů *System.Web.Mvc* obsahuje implementaci samotného MVC Frameworku. Zahrnuje však i spoustu podpůrných tříd, např. pro usnadnění práce s Ajaxem, HTML, práci s atributy nebo vytváření obsahu ve formátu JSON. *System.Web.Routing* je zodpovědná za směrování.[13] Má za úkol určit, která metoda (akce) v Controlleru má být

vyvolána na základě definované URL adresy. Framework vývojářům tímto dává plnou kontrolu nad tvorbou vlastních routovacích pravidel. Všechna tato pravidla jsou uložena v tzv routovací tabulce jako kolekce. V případě potřeby rozšíření webu např. o sekci s prefixem *Blog/*, je nutné vytvořit pravidlo (viz obr. 10) definované u ASP.NET MVC 4 v souboru *RouteConfig.cs* ve složce *App_Start*. Zde se mimo jiné nachází veškeré konfigurační třídy, které se využívají při startu aplikace.

```
routes.MapRoute(
    name: "Blog", // Název routovacího pravidla
    url: "Blog/{controller}/{action}/{id}", // Schéma URL adresy
    defaults: new {  // Výchozí hodnoty
        controller = "Blog",
        action = "Index",
        id = UrlParameter.Optional
    }
);
```

Obrázek 10: Ukázka routování – přidání vlastního pravidla

Pravidlo je vloženo do kolekce pomocí metody *MapRoute()*. Metodě se v parametrech předá unikátní název pravidla, schéma URL adresy a výchozí hodnoty, které se automaticky dosadí v případě, že URL odpovídá vytvářenému pravidlu. Vytvářet vlastní a pěkné URL adresy je tedy velmi jednoduché.

Assembly s názvem *System.Web.Abstraction* je poslední ze základních stavebních bloků ASP.NET MVC. Obsahuje soubor abstraktních tříd zapouzdřujících některé základní třídy ASP.NET infrastruktury, jež není možné nijak rozšiřovat.[13] Hlavním smyslem těchto zapouzdření je vývojářům právě toto rozšiřování umožnit.

3.4 Šablonovací systémy v rámci ASP.NET MVC

V MVC Frameworku si lze vybrat ze dvou šablonovacích systémů: Razor a ASPX. Jedná se o zásuvné moduly, které se zejména liší ve své syntaxi a jsou zodpovědné za vykreslování View v prohlížeči. Razor je novější a byl poprvé integrován až ve verzi ASP.NET MVC3.[17] Jeho příchod byl iniciován díky nutnosti kombinovat HTML přímo se zdrojovými kódy serverové části aplikace. Původní ASPX na tohle nebyl navržen. Nová syntaxe v rámci Razor nabízí lepší čitelnost kódu, podporu testování, IntelliSense (automatické dokončování kódu ve Visual Studiu) a je jednodušší na pochopení, než jeho předchůdce. Na první pohled lze tyto šablonovací systémy mezi sebou odlišit už podle přípon souborů. Razor využívá koncovku *.cshtml* nebo *.vbhtml*. Záleží na tom, jaký programovací jazyk je na straně serveru používán. ASPX využívá stejnou koncovku *.aspx* a to nezávisle na používaném jazyku.[17]

Jednou z hlavních vlastností frameworku je rozšiřitelnost a modulárnost, proto je možné rozšířit si již zabudovaný šablonovací systém, vytvořit si vlastní nebo použít dostupné alternativy jako je např. Spark, NHaml, NDjango a další. [17]

3.5 Práce s daty a Scaffolding

Součástí ASP.NET MVC Frameworku není žádná knihovna poskytující přístup k databázi. ASP.NET však dává možnost si vybrat z několika dostupných knihoven. Jednou z nich je Entity Framework, jehož tvůrcem je Microsoft. Aby byla tvorba webových stránek ještě efektivnější, ASP.NET MVC podporuje tzv. Scaffolding (česky lešení). Ten umožňuje prostřednictvím jednoduchého uživatelského rozhraní v rámci Visual Studia generovat View, Controllery i databázi na základě dat poskytovaných modelem. Scaffolding již v základu obsahuje šablony pro generování databáze a základních CRUD operací nad ní s použitím Entity Frameworku.[17] Veškeré generování kódů, souborů či složek v rámci ASP.NET MVC je založeno na tzv. T4 šablonách viz obr. 11.[17] Visual Studio poskytuje možnost vytvářet nebo upravit již existující šablony. Není nutné se tedy omezovat pouze na použití Entity Frameworku při generování kódu, ale lze si připravit šablony vlastní pro jiné databázové knihovny.

```
<#@ template language="C#" HostSpecific="True" #>
<# MvcTextTemplateHost mvcHost = (MvcTextTemplateHost)(Host); #>
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace <#= mvcHost.Namespace #>
{
    public class <#= mvcHost.ControllerName #> : Controller
    {
        //
        // GET: <#= (!String.IsNullOrEmpty(mvcHost.AreaName)) ?
          ("/" + mvcHost.AreaName) :
         String.Empty #>/<#= mvcHost.ControllerRootName #>/

        public ActionResult Index()
        {
          return View();
        }

<#
if(mvcHost.AddActionMethods) {
#>
```

Obrázek 11: Příklad části T4 šablony generující Controller

3.6 Podpora JavaScriptu

V dnešní době je JavaScript součástí každé moderní webové aplikace. Vývojáři ASP.NET MVC Frameworku si toho jsou dobře vědomi, a proto do něj integrovali řadu podpůrných knihoven, které s ním usnadní práci jak na straně klienta, tak i na straně serveru.

Pokud ASP.NET MVC projekt je vytvořen pomocí základní šablony, kterou Visual Studio nabízí, lze si všimnout automaticky vygenerované složky *Script*. ASP.NET MVC je založeno na konvencích, proto by se do této složky měly dávat veškeré JavaScriptové soubory v rámci aplikace.[13] Výchozí složka již obsahuje populární knihovnu jQuery[1] a další její rozšíření, např. k zajištění podpory validace vstupních dat na straně klienta. Součásti knihovny jQuery je i podpora asynchronního dotazování na server pomocí technologie Ajax, která umožňuje aktualizovat části stránky bez nutnosti jejího znovunačtení. MVC Framework je na toto připraven použitím tzv. *PartialView* šablon sloužících k nadefinování jen určité části stránky. Tuto částečnou šablonu lze pak používat i v klasických View nebo ji lze využít spolu s Ajaxem k aktualizaci jen některých bloků stránky. Framework nabízí pomocné třídy, které pomocí syntaxe Razor jednoduše umožní definovat Ajaxové formuláře nebo odkazy. Controller umí rozeznat, zda se jedná o Ajaxový či normální *HTTP* dotaz, čímž umožňuje adekvátně reagovat na různé typy požadavků. Například podle typu požadavku může rozhodnout, zda jako odpověď vrátí částečný View, překreslí celý View nebo pošle data ve formátu JSON.

3.7 Bezpečnost

V dnešní době je počet útoků na různé webové aplikace stále častější, proto je třeba webové stránky dostatečně chránit a předcházet tak případným problémům. MVC Framework zahrnuje spoustu prvků, které k tomu mohou dopomoci. Nejčastěji jsou tyto útoky způsobeny neošetřenými vstupními daty, např. z formulářů. Pokud je ale použita syntaxe Razor, lze se tomuto druhu útoku elegantně vyhnout, jelikož všechno, co se vyskytuje za značkou @, je automaticky HTML enkódováno.[13] Nepovolené znaky jsou převedeny na symboly, jež po vykreslení prohlížečem nijak neovlivní vzhled, či funkčnost aplikace. Veškerá vstupní data odeslána na server jsou pak znovu frameworkem prověřena a spolu s nimi i obsah případné cookie.[13] Toto je výchozí chování frameworku, které je možné ovlivnit pomocí atributů. Díky atributům lze aplikaci chránit i před různými typy útoků. Příkladem může být obrana vůči CSRF viz kód na obr. 12.

[1] Samotnou knihovnu a její dokumentaci najdete na http://www.jquery.com/

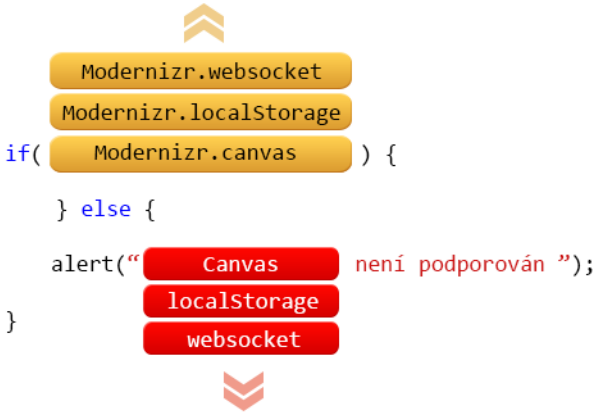
```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult ZpracujFormular()
{
    //Zpracování formuláře...
    return RedirectToAction("Index");
}
```

Obrázek 12: Obrana proti CSRF útoku

Na obrázku je zobrazena metoda Controlleru, která zpracuje příchozí *HTTP POST* požadavek nějakého formuláře. Pomocí atributu *[ValidateAntiForgeryToken]* se ověří, zda požadavek přichází z důvěryhodného zdroje a je-li tomu tak, provede se kód uvnitř metody. V opačném případě je vyvolána výjimka. Pokud bychom chtěli, aby výše zmíněnou metodu mohl vyvolat jen určitý typ uživatele, existuje atribut *[AuthorizeAttribute]*, který přesně zmíněné dělá. Tvůrci ASP.NET MVC otázku bezpečnosti určitě nepodcenili. Je však na bedrech vývojářů, jak s těmito možnostmi naloží.

4 HTML 5 v rámci ASP.NET MVC

Přestože HTML 5 ještě není oficiálně dokončeno, Microsoft již implementoval podporu jeho syntaxe v rámci Visual Studia. Pokud je používána jeho starší verzi z roku 2010, lze si doinstalovat balíček *ASP.NET MVC 3 Tools Update*[2], který obsahuje HTML 5 verze všech šablon MVC Frameworku. Jeho součástí je i balíčkový systém NUGET a různé podpůrné JavaScriptové knihovny. Příkladem může být knihovna Modernizr[3], jež je schopna během několika milisekund analyzovat schopnosti prohlížeče návštěvníka webu a detekovat tak podporu jednotlivých vlastností technologie HTML 5.[16] Výsledek analýzy se uloží do JavaScriptového objektu s názvem Modernizr.[16] Vývojáři pak jednoduše mohou upozornit uživatele, že jejich prohlížeč danou funkcionalitu nepodporuje, viz obr. 13, případně ji realizovat jiným způsobem bez použití HTML 5.



Obrázek 13: Testování podpory vlastností HTML 5 pomocí knihovny Modernizr

4.1 Klientská část

4.1.1 Formuláře a validace

Kombinace Razoru, JavaScriptových knihoven a HTML 5 může výrazně usnadnit práci při vytváření uživatelských rozhraní. V současné době se snad na všech webových stránkách lze setkat s různými typy formulářů. ASP.NET MVC nabízí několik způsobů, jak formuláře vytvářet a validovat data, která jsou jimi odeslána. Validace probíhá jak na straně klienta, tak na straně serveru.[14] Automatická validace formulářů je zapnutá v základním nastavení frameworku.[14] Nastavení lze však změnit, a validaci na straně klienta např. vypnout, pomocí konfiguračního souboru *web.config* viz obr. 14.

[2]Ke stažení na http://www.microsoft.com/en-us/download/details.aspx?id=1491
[3]http://www.modernizr.com/

```
<configuration>
  <appSettings>
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  </appSettings>
</configuration>
```

Obrázek 14: Konfigurace klientské validace

Validaci zajišťuje doplněk knihovny jQuery s pomocí tzv. Unobtrusive JavaScriptu, jehož princip je založen na tom, že se validační pravidla vyskytují přímo u jednotlivých elementů formuláře. K jejich zápisu se využívají nové datové atributy jazyka HTML 5 s prefixem *data-*, k nimž skript přistupuje a řídí podle nich samotnou validaci.[14] Validační pravidla se v ASP.NET MVC vytváří pomocí DataAnnotations atributů definovaných v modelu u jednotlivých vlastností objektu viz obr. 15. Zde je např. s použitím atributu *[Required]* definovaná povinná položka, určen formát vstupních dat atributem *[DataType]* nebo pomocí *[DisplayName]* vytvořeny uživatelsky přívětivější názvy vlastností. Na základě vytvořeného datového modelu lze pomocí Scaffolding vygenerovat kód uživatelského rozhraní (View).

Ukázka View formuláře je zobrazena na obr. 16. Pomocí syntaxe Razor a metod HTML helperů se vygenerují vstupní položky, popisky a validační zprávy formuláře. Metody např. *LabelFor()* a *EditorFor()* vytvářející výsledný HTML kód jsou natolik chytré, že díky reflexi dokáží zjistit, jaké anotace byly použity u jednotlivých vlastností modelu a automaticky vytvoří odpovídající validační HTML 5 data atributy. Ty následně využije doplněk knihovny jQuery při validaci formuláře.

Pokud by se v konfiguračním souboru klientská validace vypnula, validace na základě definovaných anotací v modelu bude dále prováděna na straně severu. Použité helpery pro generování formulářových elementů již ale nevygenerují HTML 5 data atributy, nicméně některá políčka formuláře, např. email, se stále validovat budou. Je to způsobeno tím, že metody helperů již částečně podporují nové typy HTML vstupních elementů. Při generování políčka pro email tak automaticky doplní, že vstupní pole je typu *email*. Pak už záleží na prohlížeči uživatele, zda podporuje tyto HTML 5 prvky a obstará validaci těchto polí bez použití JavaScriptu.

4.1.2 Tvorba helperů pro generování HTML 5 kódu

Hlavní smysl Razoru je ulehčit vývojářům jejich práci. Tomu napomáhají tzv. pomocníci z angl. Helpers. Jedná se o komponenty, které poskytují specifickou funkcionalitu a jsou v rámci Razoru dostupné s použitím jednoho řádku kódu. Tyto komponenty umožňují např. vygenerovat kód formuláře, odkazy, aktuální datum apod. pomocí funkcí, jež se později přeloží na HTML kód. Razor umožňuje vytvářet vlastní pomocné funkce, které

```
public class FormularModel
{
    [DisplayName("Vaše jméno:")]
    public string Jmeno { get; set; }

    [DisplayName("Vaše příjmení:")]
    [Required, StringLength(12,
        ErrorMessage="Příjmení nesmí mít více než 12
        znaků a méně než 2 znaky", MinimumLength = 2)]
    public string Prijmeni { get; set;  }

    [DisplayName("Váš email:")]
    [DataType(DataType.EmailAddress)]
    public string Email { get; set;  }

    [DisplayName("Věk:")]
    [Display(Name = "Decimal", Prompt = "Type a decimal"),
    Range(1.0, 100.0), Required]
    public decimal? Vek { get; set;  }

    [DisplayName("Vaš datum narození:")]
    [DataType(DataType.Date)]
    public DateTime DatumNarozeni { get; set;  }
}
```

Obrázek 15: Ukázka validačních pravidel v modelu

```
@Html.LabelFor(model => model.Prijmeni)

@Html.EditorFor(model => model.Prijmeni)
@Html.ValidationMessageFor(model => model.Prijmeni)
```

Kód zapsaný v razoru

Vygenerovaný kod

```
<label for="Prijmeni">Vaše příjmení</label>

<input data-val="true"
       data-val-length="Příjmení nesmí mít více než 12 znaků a méně než 2 znaky"
       data-val-length-max="12" data-val-length-min="2"
       data-val-required="Pole Vaše příjmení: je povinné."
       id="Prijmeni" name="Prijmeni" type="text" value="" />
<span data-valmsg-for="Prijmeni" data-valmsg-replace="true"></span>
```

Obrázek 16: Kód formuláře vygenerovaný na základě Razor syntaxe

se nachází v samostatných *cshtml* nebo *vbhtml* souborech, pomocí syntaxe viz ilustrace na obr. 17.

```
@helper VyvoreniCanvasElementu(string nazev, int sirka, int vyska) {
    <canvas id="@nazev.Replace(" ","_")" width="@sirka" height="@vyska">
        Tento text se zobrazí, pokud Váš prohlížeč nepodporuje element Canvas
    </canvas>

    <script>
        var canvas = document.getElementById('@nazev.Replace(" ","_")');
        var context = canvas.getContext('2d'); // Zpřístupní canvas pro kreslení
        context.fillRect(30, 30, 30, 30);  // Vykreslí čtvereček
        context.textBaseline = "top"; //  Určuje svislé zarovnání textu
        context.font = 'italic 30px sans-serif'; // Nastavení velikosti písma a fontu
        context.fillText('@nazev', 60, 30); // vykreslí text na canvas
    </script>
}
```

Obrázek 17: Pomocník pro generování HTML 5 kreslícího plátna

Soubory s helpery musí být uloženy ve složce *App_Code*. Tato složka není automaticky vygenerována v šabloně projektu, a proto je třeba ji vytvořit manuálně. Jedná se o speciální složku ASP.NET, která umožňuje používat zdrojový kód bez nutnosti kompilace, jinde by vytvoření pomocníci nefungovali.[22] Na obrázku je znázorněna ukázka pomocné funkce, která vytvoří HTML 5 element *canvas* o zadané výšce a délce. Vykreslí na něj čtverec a text zadaný v parametru metody. Funkci pak lze zavolat kdekoli v rámci šablon webového projektu pomocí Razor syntaxe ve formátu *@Nazev_cshtml_souboru .Nazev_pomocne_funkce(parametry)* a předat jí potřebné parametry, jež se doplní do kódu v jejím těle. Pomocné funkce v kombinaci s Razorem dávají možnost vytvářet čistější, udržovanější a znovupoužitelný kód, např. pro generování vlastních HTML 5 elementů, formulářových polí apod.

4.2 Serverová část

4.2.1 MVC 4, Web API a HTML 5 Web Socket

Součástí ASP.NET MVC 4 je novinka s názvem WEB API. Jedná se o framework, který do prostředí MVC 4 přináší možnost návrhu a konzumace jednoduchých HTTP služeb. Je postaven na vzorech a zvyklostech, jež jsou uplatněny v ASP.NET MVC.[17] Tímto umožňuje velmi efektivní tvorbu webových služeb, jejich správu a jednoduchou konfiguraci. WEB API je nezávislé a není striktně vázáno na ASP.NET MVC, lze ho tedy použít i ve WebForms. Některé prvky má však podobné, např. routovací systém nebo princip Controllerů. Hlavním rozdílem oproti MVC je, že Controllery ve WEB API nevracejí View, nýbrž data, nejčastěji serializovaná do podoby JSON nebo XML formátu.[17] Těchto výhod v kombinaci s MVC Frameworkem se dá využít při tvorbě aplikací založených na technologii HTML 5 Web Socket, jelikož Microsoft aktuálně integroval podporu

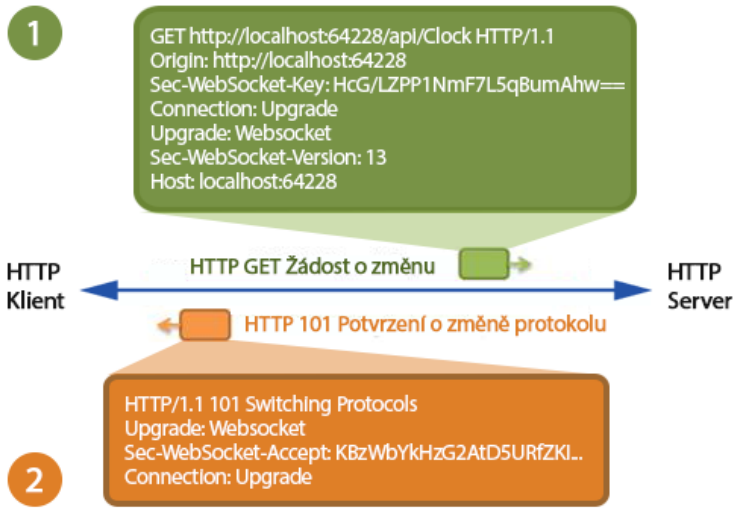
Web Socket protokolu do jeho webového serveru IIS 8 (Internet Information Server) a .NET Frameworku ve verzi 4.5. Vytvoření jednoduchého Web Socket serveru, který zasílá klientovi každé dvě sekundy informaci o aktuálním čase je na obr. 18.

```
public class ChatController : ApiController
{
    public HttpResponseMessage Get()
    {
        HttpContext.Current.AcceptWebSocketRequest(Nalsouchej);
        return Request.CreateResponse(HttpStatusCode.SwitchingProtocols);
    }

    public async Task Nalsouchej(AspNetWebSocketContext context)
    {
        WebSocket socket = context.WebSocket;
        while (true)
        {
            // Dokud je socket otevřený budeme mu zasílat aktuální čas na serveru
            if (socket.State == WebSocketState.Open)
            {
                string userMessage = "Datum: " + DateTime.Now.ToLongTimeString();
                var buffer = new ArraySegment<byte>(Encoding.UTF8.GetBytes(userMessage));
                Thread.Sleep(2000); // Uspí vlákno na 2s

                // Asynchroně zašle zprávu klientovi
                socket.SendAsync(buffer, WebSocketMessageType.Text, true, CancellationToken.None);
            }
            else { break; }
        }
    }
```

Obrázek 18: Web Socket server pro zasílání aktuálního data

Na obrázku je znázorněna třída Controlleru dědící z bázové třídy *ApiController*, čímž tvoří základ pro RESTful službu a naznačuje, že se bude pracovat pouze s daty. Metoda *Get* zajistí navázaní spojení s klientem, jenž zaslal HTTP protokolem žádost o komunikaci pomocí Web Socket protokolu. Jestliže je vše v pořádku a tuto žádost příjme, nastaví ukazatel na funkci *Naslouchej*, která bude toto spojení spravovat. Současně odpoví klientovi pomocí stavového kódu *101*, jenž ho informuje o úspěšném přechodu na komunikační protokol Web Socket. Funkce *Naslouchej* obsahuje kontext aktuálního připojení a každé dvě sekundy asynchronně zasílá zprávu klientovi o aktuálním čase, dokud spojení není ukončeno.

Na obr. 19 je ilustrace přesného postupu navázání TCP spojení mezi klientem a severem. Nejprve klient zasílá žádost formou *HTTP GET* požadavku. Jeho záhlaví obsahuje hodnotu *Upgrade*, která žádá, aby server změnil komunikační protokol na Web Socket a prohlížečem vygenerovaný náhodný klíč *Sec-WebSocket-Key*. Server po přijetí této žádosti reaguje *HTTP* odpovědí vracející transformovaný klíč *Sec-WebSocket-Accept*, čímž dochází k potvrzení, že obě strany podporují Web Socket a nastává plně duplexní komunikace pomocí TCP kanálu.



Obrázek 19: Handshake mezi klientem a serverem

4.2.2 SignalR

Další alternativou pro komunikaci v reálném čase, nejen pomocí Web Socket protokolu, je knihovna SignalR[4] pro ASP.NET. Tato knihovna funguje na straně klienta i na straně serveru a zastřešuje mezi nimi různé techniky komunikace. *„Vývoj SignalR započali dva zaměstnanci společnosti Microsoft pod Open-source licencí. Později se ho však ujal Microsoft samotný.“*[23] Hlavní předností knihovny je fakt, že je schopná realizovat komunikaci více způsoby, bez vazby na konkrétní technologii. Pokud např. server i klient podporuje Web Socket protokol, SignalR automaticky začne komunikovat pomocí něj. V opačném případě využije jinou techniku či technologii pro spojení, která požadovanou funkcionalitu obstará, bez nutnosti měnit kód aplikace. SignalR má celkem čtyři mechanizmy pro komunikaci mezi prohlížečem a serverem:

- WebSocket
- Server Sent Event
- Forever Frame
- Ajax long polling

Pouze u Web Socket se jedná o opravdu plně duplexní spojení, ostatní metody a techniky tuto komunikaci jen simulují. Knihovna SignalR není součástí MVC Frameworku, pro její použití je nutné ji doinstalovat pomocí balíčkového manažeru Nuget.[19][20]

[4]Ke stažení na http://www.signalr.net/

4.2.3 JavaScript a historie prohlížeče

Vyvíjet dynamické aplikace v ASP.NET MVC je díky velkému množství pomocných tříd, částečných View a schopnosti Controllerů reagovat na Ajax dotazy, velmi jednoduché. Použití historie prohlížeče pro navigování vpřed a zpět v rámci dynamické aplikace však představovalo problém nejen pro vývojáře, ale i pro koncové uživatele. S řešením přišlo HTML 5 a jeho History API, které běží sice na straně klienta, ale v kombinaci s možnostmi frameworku umožňuje vytvářet webové aplikace pro novější i starší prohlížeče bez podpory HTML 5. Jako ukázka může posloužit kód Controlleru na obr. 20.

```
public ActionResult Text1Partial()
{
  //Pokud se jedná o ajaxoví dataz
  if (Request.IsAjaxRequest())
  {  // Vrátí se částečný view s obsahem 1
     return PartialView("_partialObsah1");
  }  // Jinak se vrátí normální view
  return View("obsah1");
}
```

Obrázek 20: Ukázka Controlleru

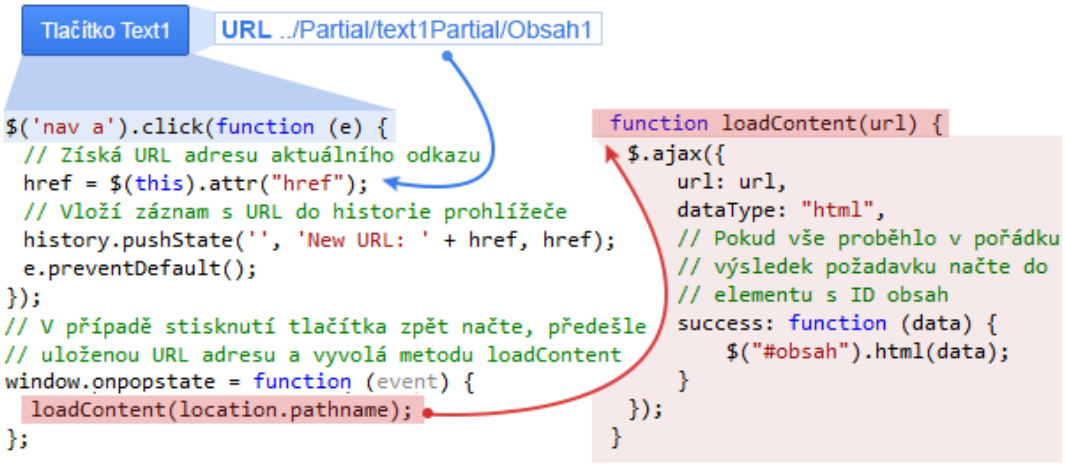
Na obrázku je metoda Controlleru vracející obsah pomocí normálního nebo částečného View v závislosti na typu požadavku. Hlavní rozdíl je v tom, že pokud se jedná o Ajax požadavek, prohlížeč nemusí stránku znovu načítat, jen dynamicky překreslí její určitou část obsahem, který se mu vrátí pomocí metody *PartialView*. HTML kód pro vyvolání zmíněné metody se vygeneruje pomocí syntaxe na obr. 21.

```
<nav>
    @Ajax.ActionLink(
        "Tlačítko Text1", //Text odkazu
        "text1Partial", "Partial", //Akce,Controller
        new { Id = "Obsah1" }, // Parametr URL adresy
        // Element ve ktrém se zobrazí výsledek
        // voláné metody
        new AjaxOptions
            {
                UpdateTargetId = "obsah"
            }
         )
</nav>
<div id="obsah"></div>
```

Obrázek 21: Ukázka View generující odkaz na metodu Controlleru

Ukázka kódu na obr. 21 vygeneruje pomocí Razor syntaxe a Ajax helperu odkaz, který v případě, že je ve stránce definován doplněk knihovny JQuery s názvem

Unobtrusive-ajax, po kliknutí vyvolá Ajax požadavek na danou akční metodu Controlleru. Mimo jiné je pomocí Helperu definován dodatečný parametr URL adresy *Id*, jenž jednoznačně identifikuje načítaný obsah. Takto vygenerovaný odkaz se pak dá použít pro práci s HTML 5 History API. V aktuální fázi se po kliknutí na jeden z odkazů dynamicky zobrazí jemu specifický obsah v *div* elementu s *id=obsah*. Tlačítko *zpět* ale v prohlížeči nefunguje, jelikož neproběhlo řádné načtení stránky, pouze změna její určité části pomocí JavaScriptu.



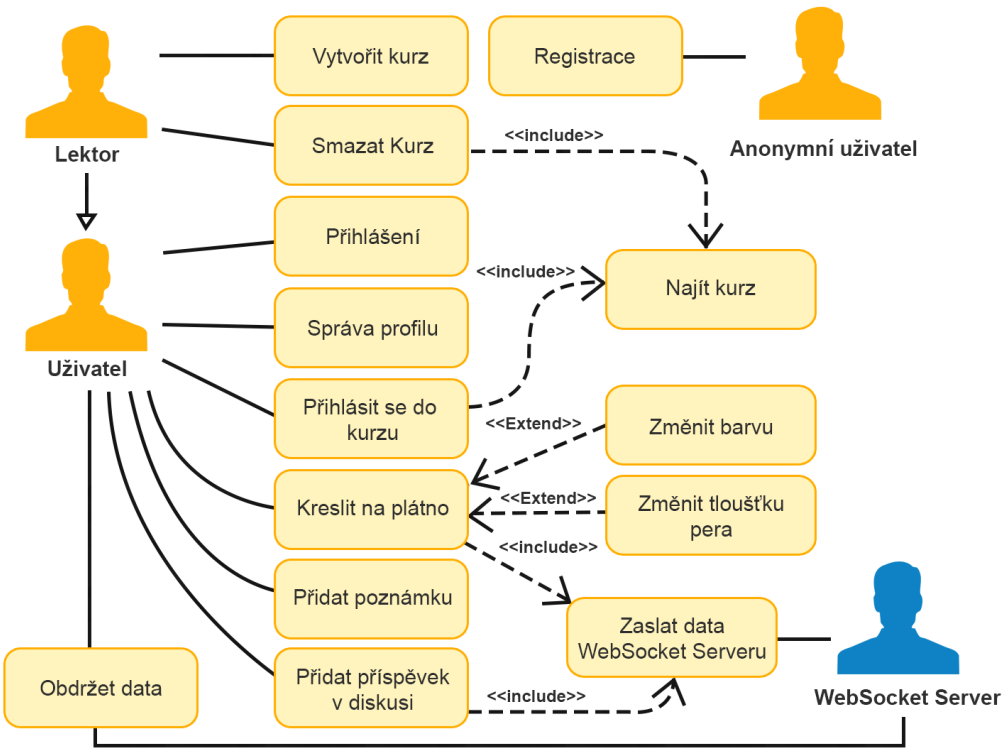
Obrázek 22: Aplikace History API

Aby historie v prohlížeči fungovala je nutné, na straně klienta, doplnit kód viz obr. 22. Při kliknutí na tlačítko se uloží pomocí HTML 5 metody *history.pushState()* URL adresa jeho odkazu do historie prohlížeče. Událost *windows.onpopstate* se pak vyvolá vždy, když uživatel klikne v prohlížeči na tlačítko *zpět* nebo *vpřed* a zavolá metodu *loadContent*, které v parametru předá uloženou URL adresu. Ta na základě poskytnuté adresy vytvoří Ajax požadavek na Controller, jenž opět vrátí požadovaný obsah, který se zobrazí uživateli. Historie prohlížeče nyní funguje. Tímto uživatel získá větší kontrolu nad JavaScriptovou aplikací.

5 Návrh a implementace ukázkové aplikace

5.1 Specifikace požadavků

Hlavním cílem této kapitoly je ilustrovat možnosti MVC 4 a vytvořit klient-server aplikaci využívající prvky technologie ASP.NET MVC v kombinaci s HTML 5. Aplikace je zaměřena na správu a provoz online kurzů, kde primární komunikační prostředek tvoří interaktivní kreslící plátno a chat. Veškerá komunikace mezi uživateli v rámci jednoho konkrétního kurzu probíhá v reálném čase, přitom každý uživatel musí být řádně registrován, aby se do kurzu mohl přihlásit. Pro založení a správu vlastního kurzu je nutné, aby uživatel byl evidován jako lektor, kterým se stane až při vyplnění profilové části uživatelského účtu. Cílem je taktéž poskytnout uživateli vyšší komfort při práci s aplikací, např. přiblížením se svým chováním klasickým desktopovým aplikacím, a to především pomocí dostupných prezentačních možností HTML 5. Aplikace by měla být zabezpečená proti nejčastějším druhům útoků a logicky strukturována tak, aby vyhledávače nebo jiné typy robotů, dokázaly porozumět prezentovanému obsahu. Bude schopna detekovat, zda prohlížeč podporuje použité technologie, v opačném případě upozorní uživatele a nabídne řešení problému. Účelem ukázkové aplikace je demonstrovat klíčové prvky obou technologií, které se dají v praxi aktuálně použít pro vývoj webových stránek. Chování systému z pohledu uživatele ilustruje diagram případu užití na obr. 23.



Obrázek 23: Diagram případů užití aplikace

5.2 Technická specifikace

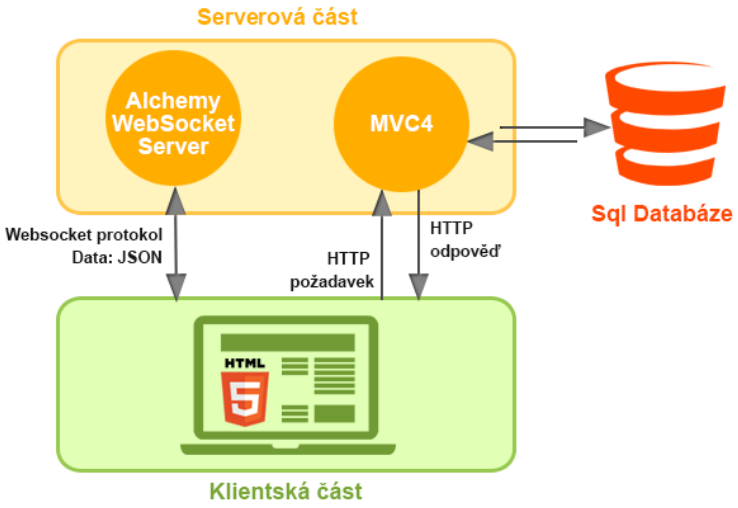
Aplikace bude rozdělena na klientskou a serverovou část. Klientská část bude využívat v rámci webového prohlížeče možnosti technologie HTML 5, JavaScript, její podpůrně knihovny a CSS 3. CSS 3 umožní definovat vzhled stránek v závislosti na použitých elementech jazyka HTML. Jelikož aktuálně existují webové prohlížeče bez podpory nebo jen s částečnou podporou vlastností HTML 5, je doporučeno využívat prohlížeče Chrome a Firefox v nejnovějších verzích se zapnutou podporou JavaScriptu. Cílem klientské části je umožnění komunikace uživatele s webovým serverem a prezentace uživatelského rozhraní.

Serverová část aplikace bude založena na technologii a principech ASP.NET MVC 4 s využitím systému Razor View engine pro tvorbu uživatelského rozhraní. Jejím hlavním úkolem bude zajištění podpory komunikačního protokolu Web Socket, který umožní plně duplexní komunikaci s klientem. Rovněž bude obsahovat aplikační logiku, zprostředkovávat komunikaci s databází a obsluhovat požadavky klientů. Aplikace bude na straně serveru využívat databázi MS-SQL Server.

5.3 Návrh řešení

Serverovou část aplikace tvoří ASP.NET MVC 4 projekt vytvořený na základě výchozí šablony s názvem Internet Apllication. Tato předpřipravená šablona již zahrnuje autentizační třídy, které umožní registraci, přihlášení a správu uživatelského účtu. Aby bylo možné s klientem komunikovat prostřednictvím protokolu Web Socket, byl použit Alchemy Web Socket server[5]. Ten se spouští jako konzolová aplikace v separátním projektu Visual Studia. Alchemy server je možné do projektu doinstalovat pomocí balíčkového systému Nuget. Architektura aplikace využívá JSON jako primární formát dat pro přenos mezi klientem a Web Socket serverem. Podpora tohoto datového formátu je na straně serveru zajištěna knihovnou JSON.NET[6] ve verzi 4.5, na straně klienta je obstarána knihovnou jQuery. Práci s databází zprostředkovává ORM Mapper Entity Framework ve verzi 5.0. Pro tvorbu databáze je použita metoda Code First, která na základě tříd a anotací v jazyce C# pomocí Entity Frameworku vygeneruje databázovou část aplikace přímo z kódu. Struktura aplikace je zobrazena na obr. 24.

[5]Ke stažení na http://www.alchemywebsockets.net/
[6]Ke stažení na http://www.json.codeplex.com/



Obrázek 24: Struktura ukázkové aplikace

U MVC aplikace se o aplikační logiku, kromě již v základu implementovaných tříd pro správu uživatelských účtů, starají dvě třídy *CourseModel* a *ProfileModel*. Základní myšlenkou je fakt, že kurz může vytvořit a spravovat pouze uživatel, který je zároveň učitelem, přitom každý kurz musí mít pouze jednoho učitele.

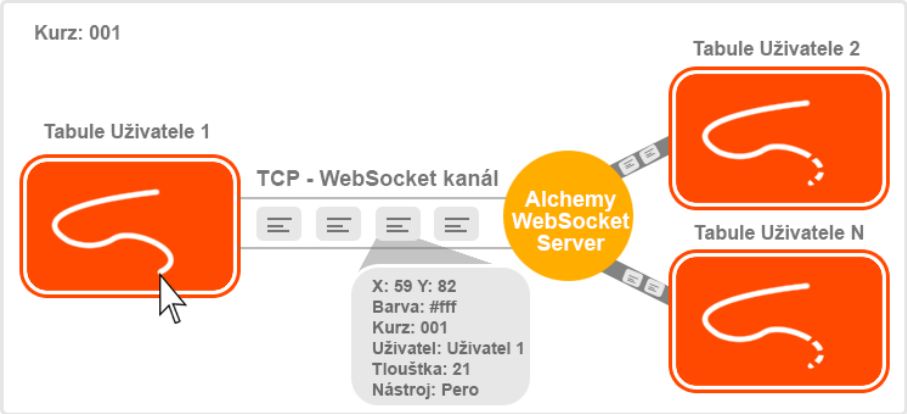
Web Socket server umožní uživatelům v rámci kurzu vzájemně komunikovat prostřednictvím diskuse, či kreslícího plátna, v reálném čase. Aby to bylo možné, musí server spravovat vlastní kolekci kurzů a v nich seznam přihlášených uživatelů. Jakmile je navázáno spojení se serverem, oba navzájem komunikují pomocí výměny zpráv ve formátu JSON. Každá zpráva obsahuje identifikaci, jaký typ informace obsahuje, aby na ni mohl klient, či server patřičně zareagovat. Jedna z prvních zpráv, jež je na server odeslána, nese informaci o jméně klienta a názvu kurzu. Jakmile ji server obdrží, zjistí, zda již daný kurz v rámci kolekce kurzů na Web Socket serveru existuje. Pokud ne, server vytvoří kurz nový a přidá do něj uživatele. Je-li spojení se serverem ukončeno, uživatel bude z kolekce automaticky odebrán.

5.4 Ukázka praktické implementace

5.4.1 Přenášení dat mezi aplikací a Web Socket serverem

Hlavní část aplikace na straně klienta tvoří kreslící plátno realizované pomocí HTML 5 elementu *Canvas*, jenž v kombinaci s dostupným API a knihovnou jQuery umožní snímat pohyb a chování myši nad plátnem. Díky tomu je možné na plátno kreslit. Zmíněných možností je využito i pro tvorbu vzorníku barev. Ten umožňuje uživateli jednoduše měnit barvu štětce, kterým se bude kreslit na plátno. Nastavení tloušťky štětce zajišťuje HTML 5 element *input* typu *range*, který zobrazí posuvník nebo *InputBox* v případě, že prohlížeč tento prvek nepodporuje. Informace o nastavených parametrech jednotlivých

komponent kreslícího plátna se při kreslení odesílají na Web Socket server. Ten informace přepošle ostatním klientům v rámci probíhajícího kurzu. Princip komunikace mezi tabulí a Web Socket serverem je na obrázku 25. Podobně je vyřešena i diskuse mezi uživateli, odesílaná data mají jen jiný formát a typ zprávy.



Obrázek 25: Princip HTML 5 tabule

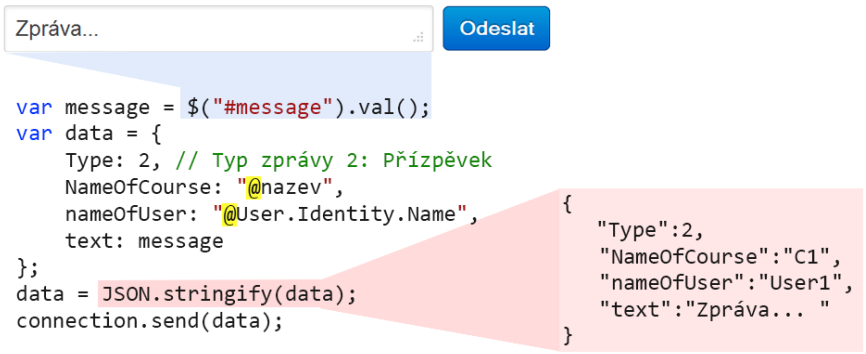
Aby bylo možné s Web Socket serverem komunikovat, klient musí komunikaci iniciovat. Na obr. 26 je vidět část zdrojového kódu metody, který je potřebný pro navázání spojení se serverem. Tato metoda se vyskytuje v Razor helperu *ClassBoard* obsahující JavaScriptový i HTML kód pro vytvoření kreslícího plátna a diskuse. Metoda současně obsahuje registrace událostí, které mohou nastat při komunikaci se serverem. Příchozí komunikaci zachycuje událost *OnMessage*. Jejím hlavním úkolem je deserializace zasílané JSON struktury a následné rozpoznání typu zprávy. Na jeho základě je provedena reakce u klienta, např. zobrazena nová zpráva v diskusi nebo kreslení na plátno.

Při kreslení nebo psaní zpráv do diskuse klient potřebuje data serveru odesílat. To umožní vytvořený globální *connection* objekt pomocí metody *send*. Odesílaná data jsou serializována do formátu JSON a předána jako parametr metody. Tuto funkčnost ilustruje zdrojový kód pro zasílání zpráv v diskusi na obr. 27. Jak je z ukázky patrné, je zde využito možnosti syntaxe Razor pro identifikaci kurzu a žáka. Tyto proměnné jsou Razor helperu předány jako parametr při jeho volání a slouží serveru jako podklad pro vyhledání běžícího kurzu v rámci vytvořené kolekce kurzů.

Jakmile Web Socket server zprávu na obr. 27 příjme, rozpozná se typ zprávy a vyvolá se na jeho základě metoda *ChatMessage*, které jsou předána zaslaná data a navíc kontext aktuálně připojeného uživatele v parametru viz obr. 28. Metoda poté nalezne požadovaný kurz a rozešle zprávu všem jeho uživatelům pomocí funkce *BroadcastMessage*.

```
var connection = {};
function openConnection() {
    if (Modernizr.websockets) {
        if (connection.readyState === undefined || connection.readyState > 1) {
            connection = new WebSocket('ws://localhost:8100');
            connection.onopen = function () {
                $("#ServerStatus").removeClass('alert');
                $("#ServerStatus").addClass('success').text("Server Online");
            };
            connection.onmessage = function (event) {
                try {  // Deserializace přijatých dat
                    var jsonData = JSON.parse(event.data);
                } catch (e) {
                    console.log('Nevalidní JSON:  ', message.data);
                    return;
                }
                if (jsonData.Type == 9) {  // Typ záprávy 9: Kreslení na plátno
                    var coordinates = jsonData.Data;
                    for (var i = 0; i < coordinates.length; i++) {
                        var current = coordinates[i].split(",");
                        var x = current[0];
                        var y = current[1];
                        context.moveTo(lastXServer, lastYServer);
                        context.lineTo(x, y);
                        context.stroke();
                        lastXServer = x;
                        lastYServer = y;
                    }
```

Obrázek 26: Vytvoření spojení s Web Socket serverem a registrace událostí



Obrázek 27: Odeslání zprávy Web Socket serveru v diskusi

```
private static void ChatMessage(string nameOfCourse, string nameOfUser,
    string text, UserContext context)
{
  var course = OnlineCourses.Keys.Single(o => o.CourseName == nameOfCourse);

  if (course.IsUserIn(nameOfUser) == false)
  {
    context.Send("Musíte byt přihlášen/a, aby jste mohl/a zaslat zprávu!");
  }
  else
  {
    course.BroadcastMessage(nameOfUser, text, context.ClientAddress);
  }
}
```

Obrázek 28: Přeposlání zprávy všem přihlášeným uživatelům v kurzu

5.4.2 Upload fotografie

O vytvoření nového kurzu v rámci MVC 4 aplikace se stará formulář, jehož součástí je i možnost přidat ke každému kurzu fotografii. Formulář je vytvořen na základě definovaných vlastností a DataAnotations atributů v modelu *CourseModel*, který reprezentuje samotný kurz. Pro perzistenci definovaného modelu do databáze je využit Entity Framework. Fotografie je nahrána na sever jejím přetažením, např. z plochy na vyznačenou oblast v rámci webové stránky. Při nahrávání je automaticky vytvořen náhled fotografie a zachycen jeho průběh. Pokud je nahrávání úspěšné, je rovněž zobrazen odkaz na uložený obrázek. Proces nahrávání nastává při vyvolání HTML 5 události *drop*, která je spuštěna při přetažení obrázku na element s třídou *uploadArea* viz obr. 29. Událost v sobě nese informace o přetažených souborech. Díky HTML 5 FileAPI lze k těmto informacím přistupovat pomocí položky *dataTransfer.files*, která obsahuje kolekci objektů *file*. Kolekce přetažených souborů je následně předána metodě *upload*. Ta vytvoří náhledový obrázek a pomocí API XmlHTTPRequest 2 odešle soubor na server.

HTML 5 FileAPI definuje interface *FileReader* viz obr. 30. Ten na straně klienta umožňuje asynchronně načítat soubory do paměti. Taktéž obsahuje sadu událostí, jež umožňují sledovat průběh načítání souboru. Jakmile je soubor načten, je vyvolána událost *onLoadEnd*, jejíž atribut *result* následně poskytne přístup k nahranému souboru. Soubor může být načten více způsoby, jako text, binární řetězec nebo jako datové url, které umožní, aby přenesená binární data ve formátu *base64* byla součástí URL adresy. Tento způsob načítání je využit pro vytvoření náhledového obrázku.

Soubor je zaslán na server asynchronně pomocí API rozhraní XMLHttpRequest 2 (XHR2). To definuje podobné události jako HTML 5 FileAPI a taktéž je u něj možné monitorovat průběh nahrávání souboru. XHR2 používá pro zaslání souboru ke zpracování na server HTTP protokol, jehož tělo je naplněno parametrem metody *send* viz obr. 31.

```
$('.uploadArea').on('drop', function (e) {
    if (e.originalEvent.dataTransfer) {
        if (e.originalEvent.dataTransfer.files.length) {

            e.preventDefault();
            e.stopPropagation();

            $("#result").fadeIn('slow');

            upload(e.originalEvent.dataTransfer.files);
        }
    }
}
```

Obrázek 29: Událost drop, která spouští nahrávání souboru na server

```
function upload(files) {
    var fileType = files[0].type;
    if (fileType == 'image/png' || fileType == 'image/jpeg') {
        // HTML5 fileReader
        var reader = new FileReader();
        // Jakmile je obrázek načten, je vytvořen jeho náhled
        reader.onloadend = function (e) {
            var img = document.getElementById("preview");
            var url = e.target.result;
            img.src = url;
        };
        // Spustí asynchroní načtení obrázku
        reader.readAsDataURL(files[0]);
        uploadImage(files[0]);
    }
    else {
        alert("Nahrát můžete poze obrázky ve formátu JPG a PNG !");
    }
}
```

Obrázek 30: Použití HTML 5 FileReader API pro vytvoření náhledového obrázku

```
var uploadImage = function (file) {
    var actualpath = window.location.pathname;
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "/Course/Upload", true);
    // Nastavení potřebných HTTP hlaviček
    xhr.setRequestHeader("Content-Type", "multipart/form-data");
    xhr.setRequestHeader("X-File-Name", file.name);
    xhr.setRequestHeader("X-File-Size", file.size);
    xhr.setRequestHeader("X-File-Type", file.type);
    xhr.setRequestHeader("X-File-Loc", actualpath);
    // Odešle soubor
    xhr.send(file);
};
```

Obrázek 31: Odeslání souboru na server pomocí XHR2

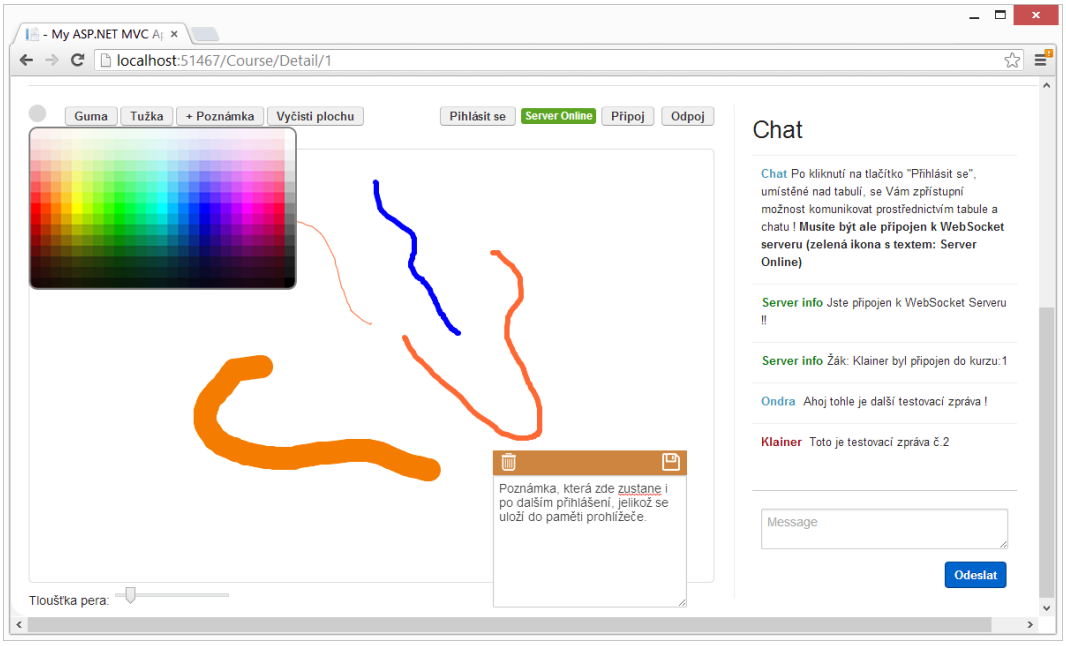
V ASP.NET MVC aplikaci tento XHR požadavek vyvolá metodu *Upload* v Controlleru *Course*. Metoda je označena atributem *[HttpPost]*, jenž zajistí možnost jejího vyvolání pouze HTTP POST požadavkem. Přístup ke kontextu aktuálního HTTP požadavku se dá získat voláním *System.Web.HttpContext.Current*, které vrací objekt typu *HttpContext*. Výše uvedené ilustruje metoda na obr. 40. Z kontextu příchozího požadavku vytvoří *inputStream*. Získaná data se uloží do objektu *file* reprezentujícího nahraný soubor, který je poté uložen do složky na straně serveru metodou *SaveFile*. Jako výsledek je vrácena cesta k nahranému souboru.

```
[HttpPost]
public string Upload()
{
  UploadFile upload = new UploadFile();
  FileToUpload file = upload.GetFile(System.Web.HttpContext.Current);
  string pathToFile = string.Empty;

  if (file != null)
  {
    //Uloží soubor na server a vrátí jeho URL adresu
    pathToFile = upload.SaveFile(file, "Course",
                System.Web.HttpContext.Current, User.Identity.Name);
  }
  return pathToFile;
 }
```

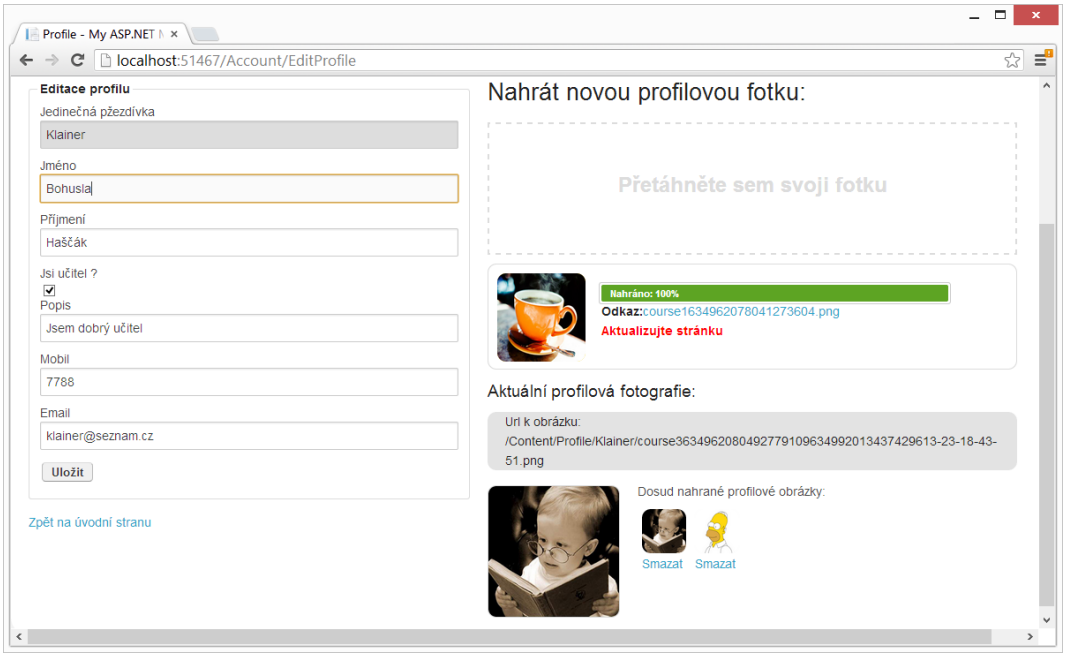
Obrázek 32: Zpracování požadavku na nahrání souboru

5.5 Výsledná aplikace

Pro ukázku je na obrázcích 33 a 34 ilustrováno uživatelské prostředí některých komponent výsledné aplikace.



Obrázek 33: Kreslící plátno v kurzu



Obrázek 34: Upload souborů pomocí drag and drop

6 Zhodnocení práce

Standard HTML 5, byť ještě není dokončen, nabízí vývojářům další alternativu jak vytvářet bohaté internetové aplikace (RIA). Internet je již nyní zahlcen spoustou aplikací a her využívajících tuto technologii, a to hlavně díky podpoře velkých korporací jako je např. Facebook, Google či Microsoft. Tito internetoví giganti do jisté míry mohou za to, že vývoj webových stránek jde tak rychle kupředu. Facebook např. využívá HTML 5 jako platformu pro vývoj online her v rámci jeho sociální sítě, Google ho integroval do poštovního klienta Gmail a zároveň zajistil jeho podporu i ve svém webovém prohlížeči Chrome, který patří mezi celosvětově nejpoužívanější. Microsoft, kromě toho že má vlastní webový prohlížeč s podporou HTML 5, integruje tento nový standard i do jeho vývojářských nástrojů jako je např. Visual Studio. Rovněž si lze podpory HTML 5 všimnout u jeho technologických platforem, např. ASP.NET MVC, kterému se tato práce věnuje nebo aktuálně v operačního systému Windows 8.

Vzájemné propojení HTML 5 a ASP.MVC má určitě smysl, což se v práci a ukázkové aplikaci především projevilo na otázkách kolem:

- bezpečnosti (validace formulářů)
- nového komunikačního protokolu WebSocket (sdílená tabule, chat)
- zvýšení uživatelského komfortu (funkční historie u JavaScriptových aplikací, Canvas, SVG, Drag and Drop - upload obrázků, . . .)
- dostupných možností vývojářských nástrojů společnosti Microsoft
- využití dalších navazujících technologií v této oblasti (SignalR, Modernizr, jQuery, Alchemy Web Sockets, . . .)

6.1 Problémy při vývoji

Zvolené řešení ukázkové aplikace, postavené na využití externí knihovny Alchemy Web Sockets pro podporu komunikace prostřednictvím Web Socket protokolu, je jedno z možných. Existují i jiné možnosti např. použití nové verze ASP.NET 4.5 v kombinaci s ISS 8 nebo knihovna SignalR. Tu bych nyní i sám použil, jelikož by vyřešila problémy související s nasazením aplikace. Právě díky použití knihovny Alchemy není aplikace dostupná z internetu a musí být hostována přímo na fyzickém zařízení. Alchemy server se totiž spouští jako konzolová aplikace pro jejíž nasazení je nutné mít vlastní VPS, případně vhodnou variantu CLOUD hostingu.

Jelikož je HTML 5 stále ve vývoji a dostupné prohlížeče nepodporují všechny jeho možnosti, objevil se problém s kompatibilitou aplikace. Prohlížeč Firefox ve verzi 19.0.2 např. nevykresluje uživateli posuvník pro změnu tloušťky pera, kterým kreslí na plátno. Tento problém je již vyřešen v testovací verzi prohlížeče pod názvem Firefox Nightly.

Další oficiální verze by již posuvník měla zobrazovat korektně. Na základě těchto zkušeností je doporučeno aplikaci používat v prohlížeči Chrome ve verzi 26 nebo IE 10. Taktéž byl při vývoji objeven problém, kdy při spuštění aplikace v IE 10 jí prohlížeč zakázal přistoupit k lokálnímu úložišti. Tento problém byl vyřešen spuštěním prohlížeče v režimu s vyššími uživatelskými právy. Jednou z příčin těchto problémů s kompatibilitou je, že prohlížeče nepoužívají stejné vykreslovací jádra. Vzhled aplikace se tedy může mírně lišit v závislosti na použitém prohlížeči.

Další problémy, hlavně ze začátku, se spíše týkaly věcí spojených s malými zkušenostmi v oblasti použití těchto technologií. Postupem času, hlavně díky kvalitně zpracovaným dokumentacím, jsem ale úspěšně dospěl k jejich řešení.

7 Závěr

V této práci jsem se věnoval popisu technologií HTML 5 a ASP.NET MVC 4, které aktuálně představují základ pro webové aplikace nové generace postavené na platformách společnosti Microsoft. Rovněž jsem se na praktických ukázkách u jednotlivých technologií snažil ilustrovat jejich specifické funkční a prezentační vlastnosti. Vzájemné propojení těchto vlastností následně vedlo k vytvoření ukázkové aplikace, která kombinuje prvky obou technologií.

Na ukázkové aplikaci se podařilo ilustrovat možnosti Web Socket protokolu s využitím Alchemy Web Socket serveru. V práci ovšem byly uvedeny i alternativní možnosti jeho implementace pomocí knihovny SignalR nebo nativní podpory v rámci .NET Frameworku ve verzi 4.5. Rovněž se, primárně díky Razoru, podařilo jednoduše vytvořit uživatelsky přívětivé prostředí aplikace a zajistit její bezpečnost proti nejčastějším typům útoků. Na základě zmapovaných klíčových možností HTML 5 v úvodu práce se do aplikace povedlo integrovat netradiční prvky, např. vzorník barev, online sdílená tabule nebo možnost psaní rychlých poznámek aj. Výsledkem propojení těchto technologií vznikla aplikace, která v souladu se zadáním umožňuje správu a provoz online kurzů.

Realizaci práce doplňuje velké množství vlastních ilustrací, jež se snaží zachytit skutečný stav problematiky, kterou jsem řešil.

8 Reference

[1] FAQ: WHATWG Wiki. WHATWG. WHATWG Wiki [online]. 2013, 23 February 2013 [cit. 2013-03-31]. Dostupné z: http://wiki.whatwg.org/wiki/FAQ

[2] NIWA, Ryosuke. The History and the Future of HTML. In: R. Niwa [online]. 2010 [cit. 2013-03-31]. Dostupné z: https://rniwa.com/2010-03-29/the-history-and-the-future-of-html/

[3] HTML5. In: Wikipedia: the free encyclopedia [online]. 2013, 31 March 2013 [cit. 2013-03-31]. Dostupné z: http://en.wikipedia.org/wiki/HTML5

[4] GOLDSTEIN, Alexis, Louis LAZARIS a Estelle WEYL. HTML5 a CSS3 pro webové designéry. Vyd. 1. Brno: Zoner Press, 2011, 286 s. Encyklopedie webdesignera. ISBN 978-80-7413-166-0.

[5] Introduction to HTML5 - HTML: MDN. In: Mozilla Developer Network [online]. 2012, Dec 14, 2012 [cit. 2013-03-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/HTML/HTML5/Introduction_to_HTML5

[6] HTML5 Web Storage loophole can be abused to fill hard disks with junk data. In: CONSTANTIN, Lucian. Computerworld [online]. March 1, 2013 [cit. 2013-04-22]. Dostupné z: http://www.computerworld.com/s/article/9237259

[7] LUBBERS, Peter, Brian ALBERS, Frank SALIM a Tony PYE. Pro HTML5 programming. 2nd ed. New York: Apress, c2011, xx, 332 p. Expert's voice in Web development. ISBN 978-1-4302-3864-5.

[8] Server-Sent Events. In: World Wide Web Consortium (W3C) [online]. Ian Hickson. 11 December 2012 [cit. 2013-03-12]. Dostupné z: http://www.w3.org/TR/2012/CR-eventsource-20121211/

[9] DOM Storage: Document Object Model (DOM). In: Mozilla Developer Network [online]. Mar 9, 2013 [cit. 2013-03-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/DOM/Storage

[10] IndexedDB: MDN. In: Mozilla Developer Network [online]. 2013, Feb 28, 2013 [cit. 2013-04-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/IndexedDB

[11] JACKSON, Jim a Ian GILMAN. HTML5 for .NET developers: single page web apps, JavaScript and semantic markup. Shelter Islands, NY: Manning, 2013, xxv, 388 p. ISBN 16-172-9043-2.

[12] MACKEY, Alex. Introducing .NET 4.0 with Visual studio 2010. New York: Distributed to the book trade worldwide by Springer Verlag, c2010, xxxi, 471 p. Expert's voice in .NET. ISBN 9781430224563-.

[13] GALLOWAY, Jon S. Professional asp.net mvc 3.0. 1st ed. Indianapolis, IN: Wiley Pubishing, Inc., 2011, p. cm. ISBN 11-180-7658-3.

[14] FREEMAN, Adam. Pro ASP.NET MVC 3 framework. 3rd. ed. New York: Apress, c2011. The expert's Voice in.NET. ISBN 978-1-4302-3404-3.

[15] ASP.NET MVC: Overview. MSDN: the Microsoft Developer Network [online]. [cit. 2013-03-28]. Dostupné z: http://msdn.microsoft.com/en-us/library/dd381412

[16] Modernizr Documentation. Modernizr: the feature detection library for HTML5/CSS3 [online]. © 2009-2013 [cit. 2013-04-23]. Dostupné z: http://modernizr.com/docs/

[17] PALERMO, Jeffrey. ASP.NET MVC 4 in action. Rev. ed. Shelter Island, NY.: Manning, 2012, xxxii, 406 p. ISBN 16-172-9041-6.

[18] PETRICEK, Tomas. .NET: Úvod do Reflection. In: Vyvojar.cz [online]. 19.6.2005 [cit. 2013-04-18]. Dostupné z: http://www.vyvojar.cz/Articles/307-net-uvod-do-reflection.aspx

[19] Transports and Fallbacks. In: FLETCHER, Patrick. The Official Microsoft ASP.NET Site [online]. February 27, 2013 [cit. 2013-03-31]. Dostupné z: http://www.asp.net/signalr/overview/introduction/transports-and-fallbacks

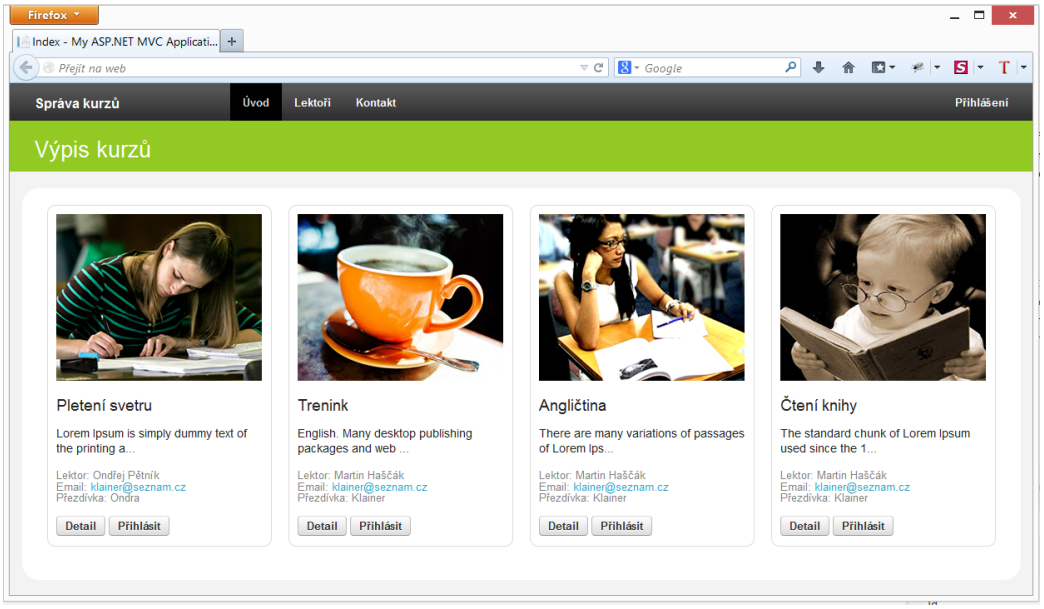
[20] CHADWICK, Jess, Todd SNYDER a Hrusikesh PANDA. Programming ASP.NET MVC 4. 1st ed. Sebastopol, CA: O'Reilly Media, c2012, xvi, 471 p. ISBN 14-493-2031-7.

[21] BALLIAUW, Maarten a Xavier DECOSTER. Pro NuGet. New York: Springer Science Business Media (worldwide distributor), c2012, xxvi, 285 p. ISBN 978-143-0241-928.

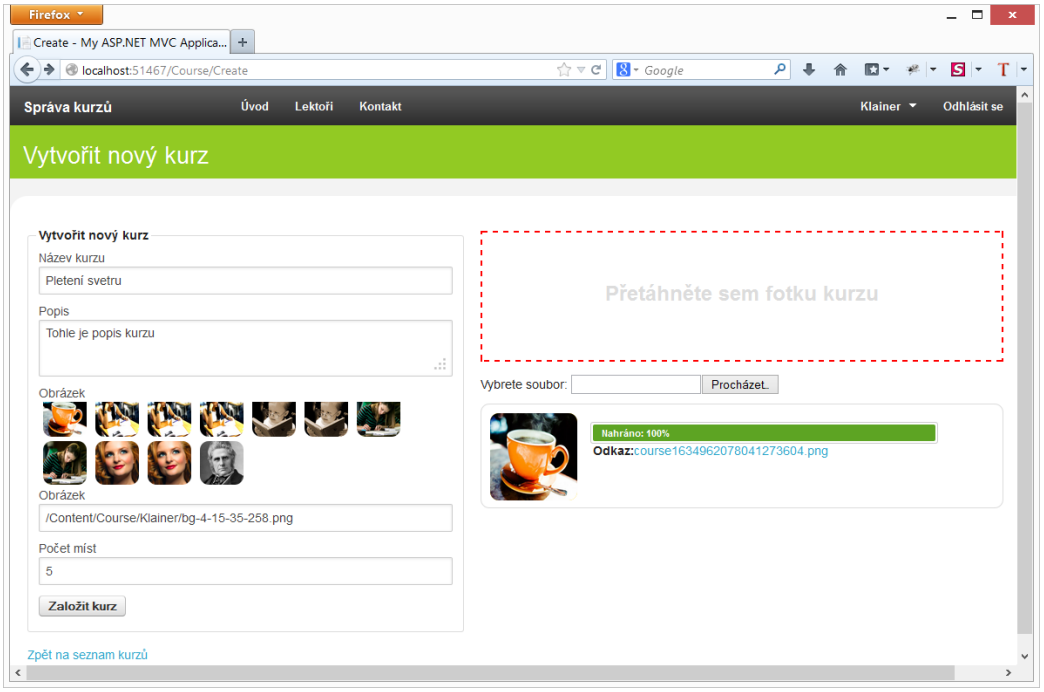
[22] ESPOSITO, Dino. Working with ASP.NET MVC 3 Razor Helpers and Templates. In: Dev Pro [online]. Jul. 12, 2011 [cit. 2013-04-23]. Dostupné z: http://devproconnections.com/aspnet-mvc/working-aspnet-mvc-3-razor-helpers-and-templates

[23] MOHL, Daniel. Building web, cloud, and mobile solutions with F# [online]. 1st ed. Sebastopol, CA: O'Reilly, 2012, xii, 160 p. [cit. 2013-04-18]. ISBN 14-493-3376-1. Dostupné z: http://books.google.cz/books?id=dZrslEoAb58C. Strana 105.

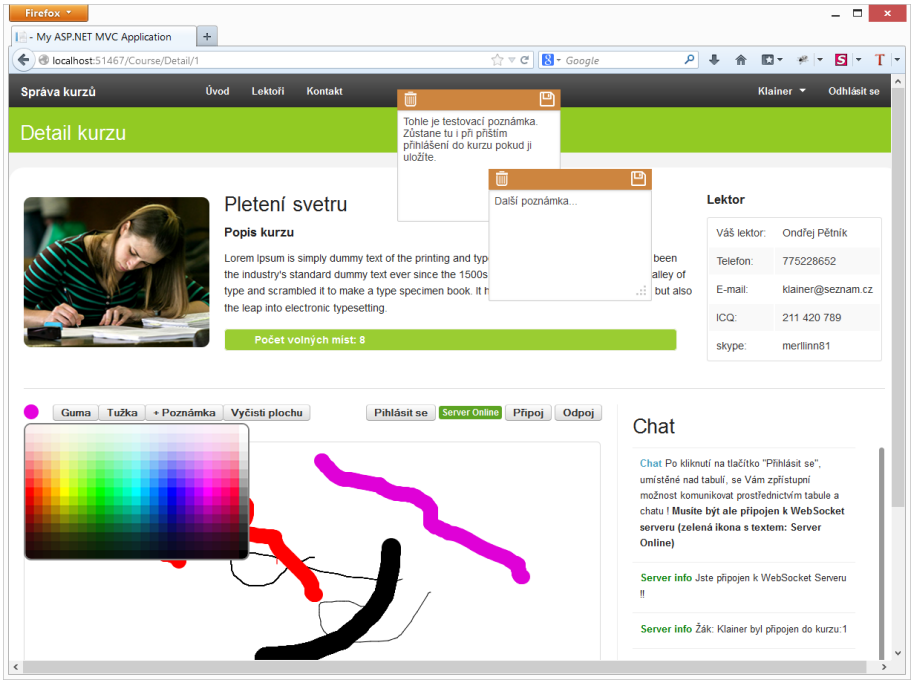
A Ukázka uživatelského rozhraní aplikace



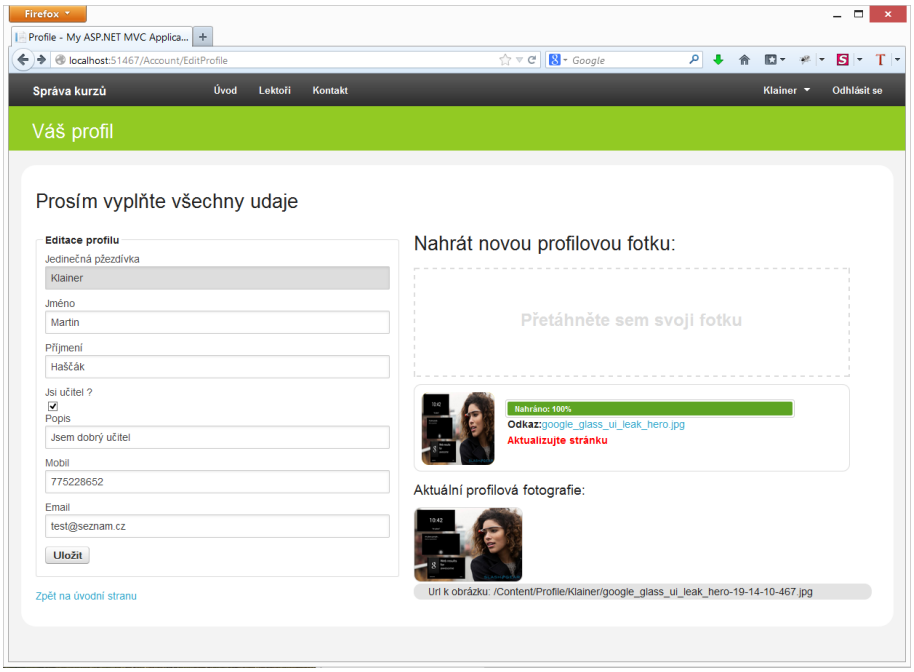
Obrázek 35: Úvodní obrazovka aplikace s výpisem kurzů



Obrázek 36: Vytváření nového kurzu - nahrávání fotografií

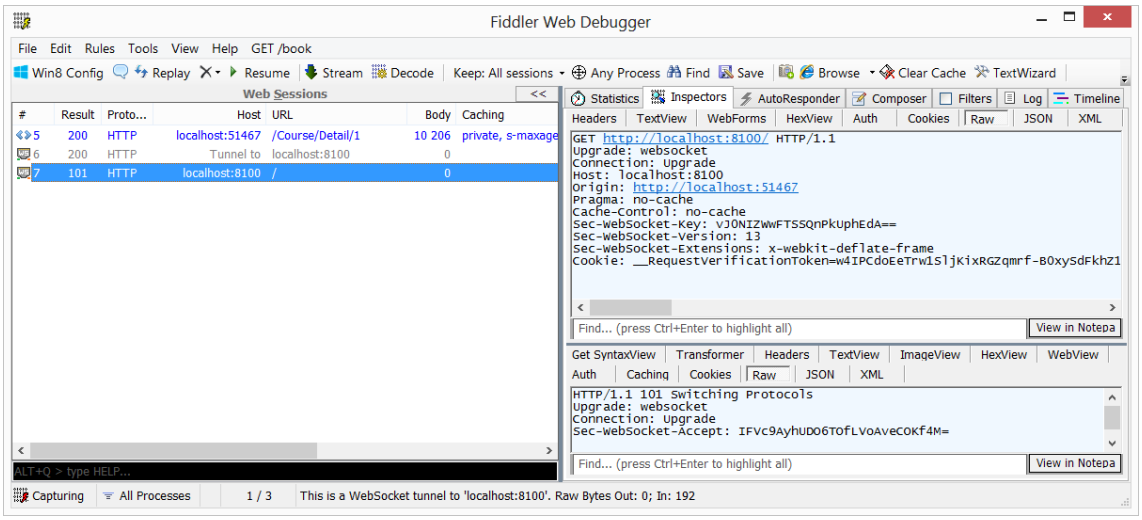


Obrázek 37: Detail kurzu - kreslící plátno, vzorník barev, diskuse, poznámky

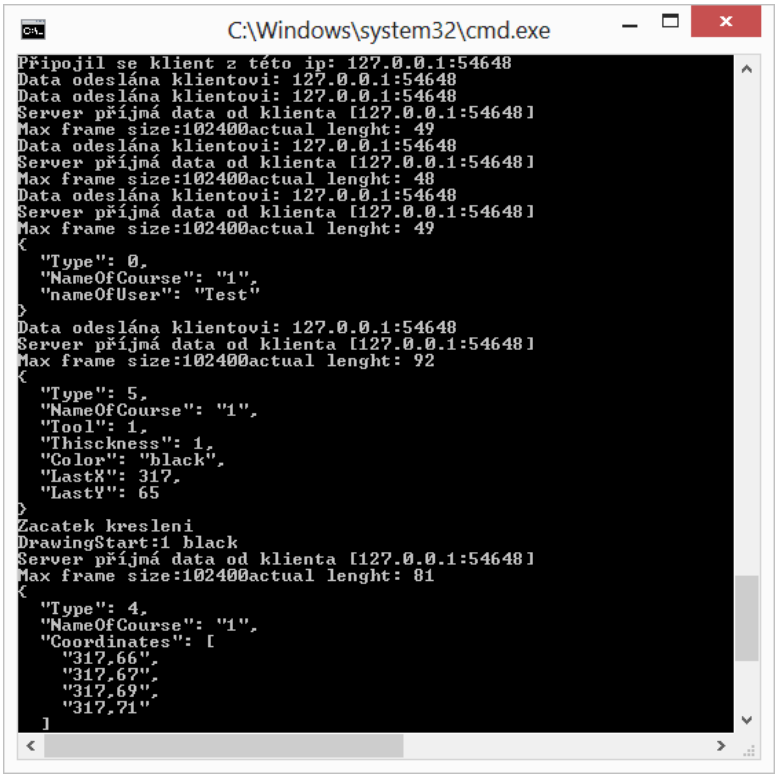


Obrázek 38: Správa profilu uživatele

B Ukázka komunikace pomocí Web Socket



Obrázek 39: Zachycení změny komunikačního protokolu pomocí aplikace Fiddler



Obrázek 40: Záznam komunikace mezi aplikací a Alchemy Web Socket Serverem

C Obsah přiloženého CD

Na přiloženém CD se nachází kompletní zdrojový kód aplikace. Rovněž se zde nachází ukázkové video demonstrující provoz aplikace. V následující tabulce je uvedena struktura přiloženého CD.

Adresář	**Popis**
/Src/	Zdrojové kódy aplikace
/Video/	Ukázkové video demonstrující provoz aplikace
/Doc/	Krátký návod pro spuštění aplikace
/Text/	Kompletní text práce